BLM6112
Advanced Computer Architecture

Memory Hierarchy

Prof. Dr. Nizamettin AYDIN

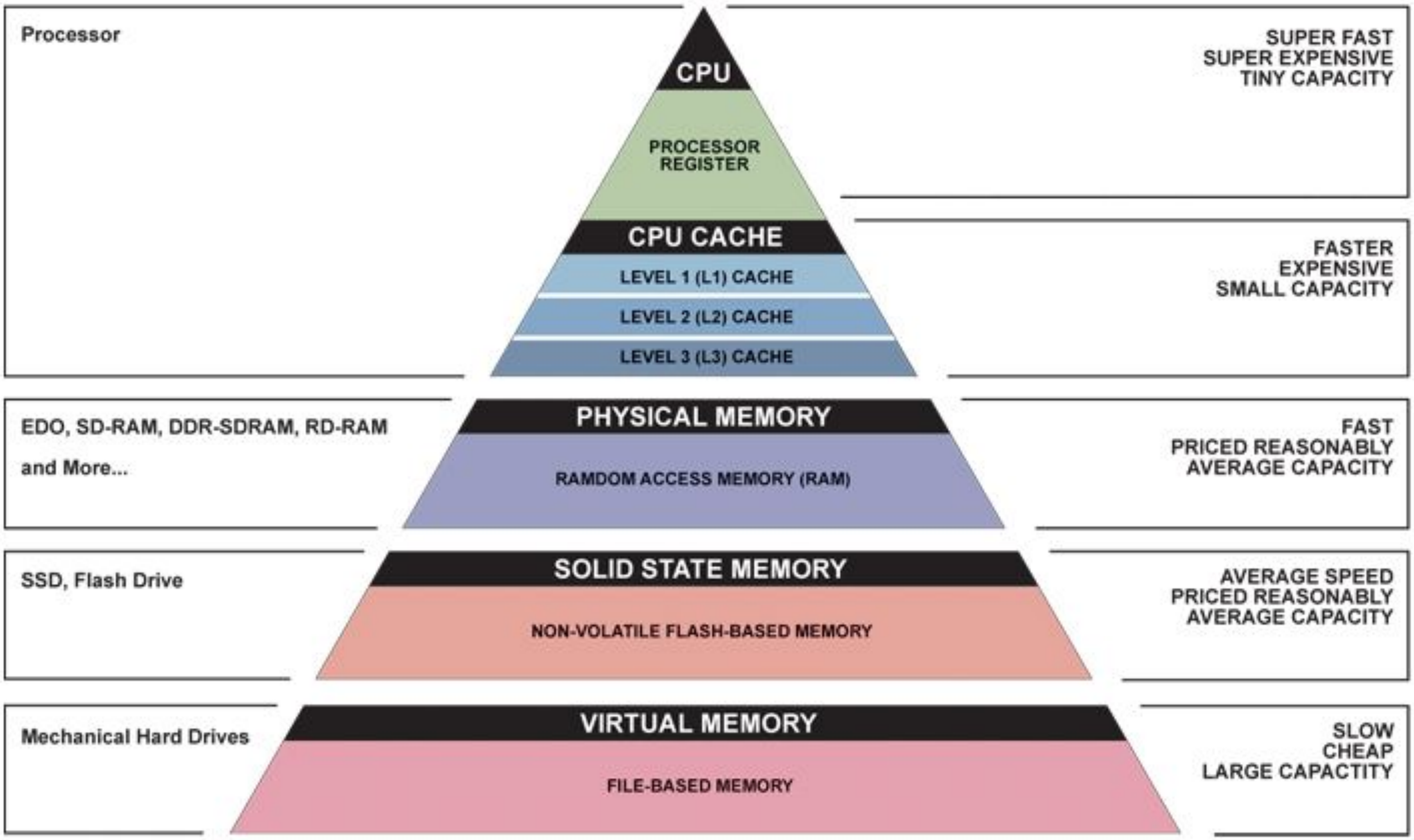
naydin@yildiz.edu.tr

http://www3.yildiz.edu.tr/~naydin

Outline

- Introduction
- Memory Hierarchy
- Cache Memory
- Cache Performance
- Main Memory
- Virtual Memory
- Translation Lookaside Buffer
- MIPS R4000 Case Study

Computer Memory Hierarchy



Introduction

- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution:
  - organize memory system into a hierarchy
    - Entire addressable memory space available in largest, slowest memory
    - Incrementally smaller and faster memories,
      - each containing a subset of the memory below it, proceed in steps up toward the processor
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the allusion of a large, fast memory being presented to the processor

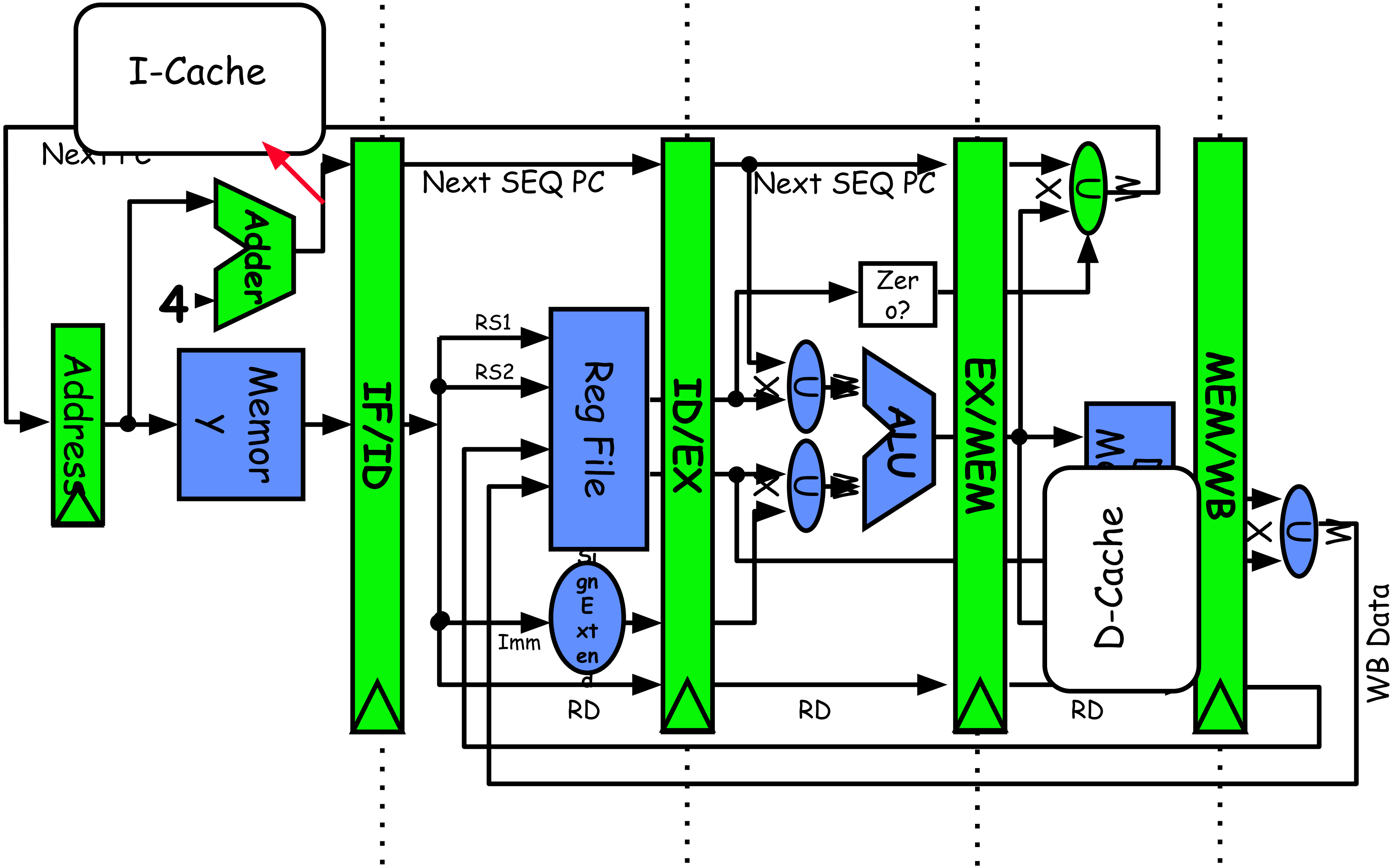
The Principle of Locality

- The Principle of Locality:
  - Programs access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time):
    - If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space):
    - If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)

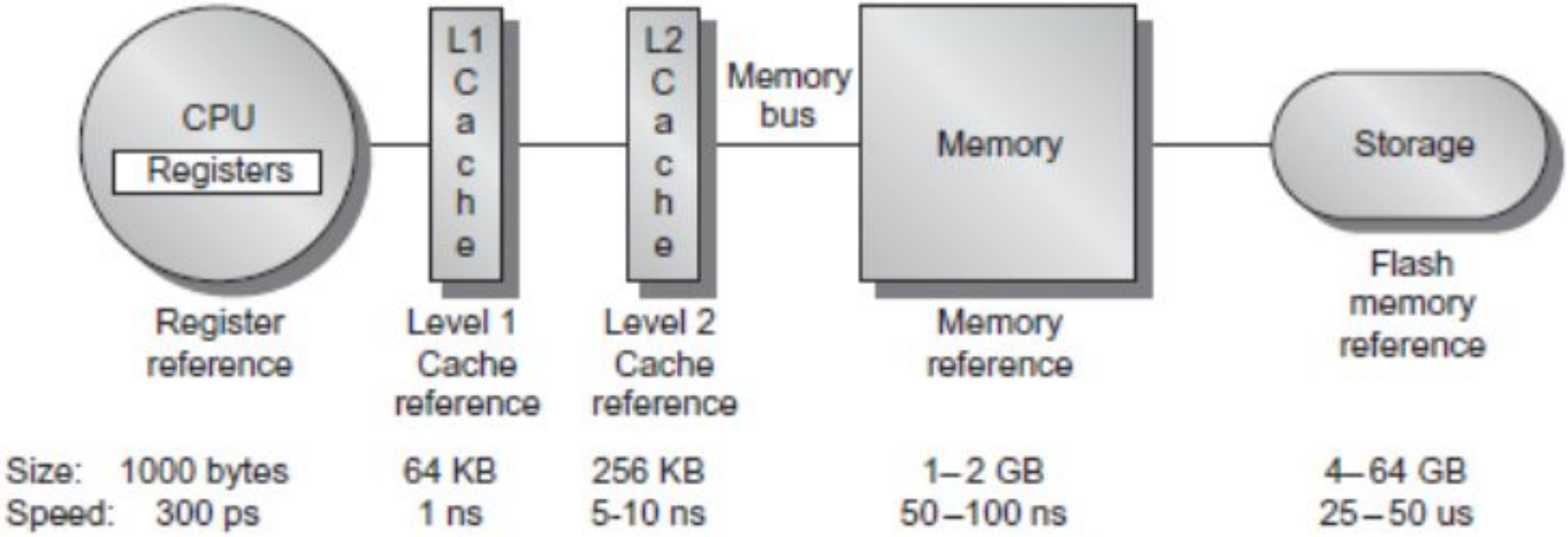
What is a Cache?

- Small, fast storage used to improve average access time to slow memory.
- Exploits spatial and temporal locality
- In computer architecture, almost everything is a cache!
  - Registers "a cache" on variables – software managed
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)
  - TLB a cache on page table
    - TLB:translation lookaside buffer
  - Branch-prediction a cache on prediction information?
  - Gives the allusion of a large, fast memory being presented to the processor

Cache and Pipelining

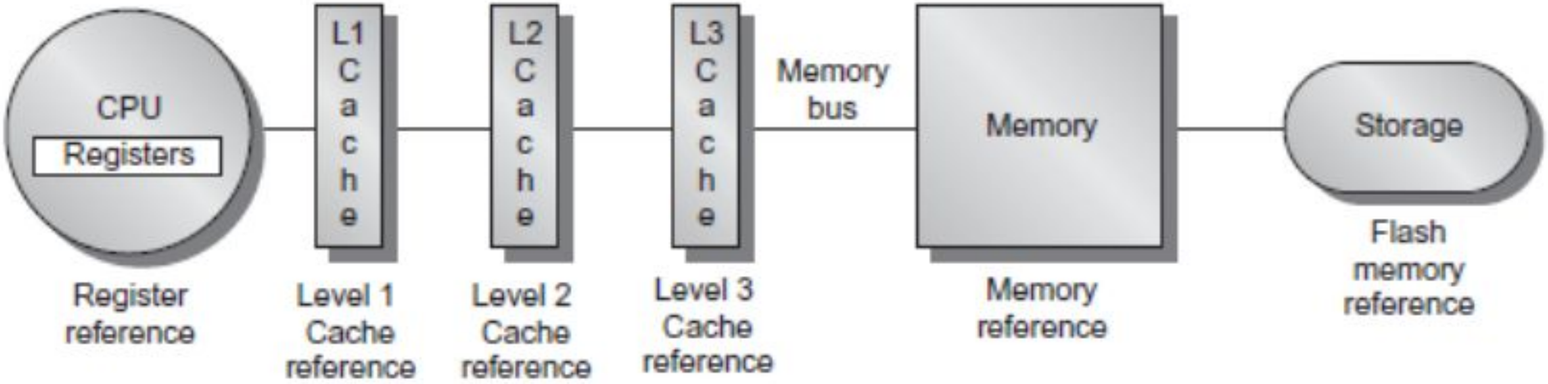


Memory Hierarchy – PMD



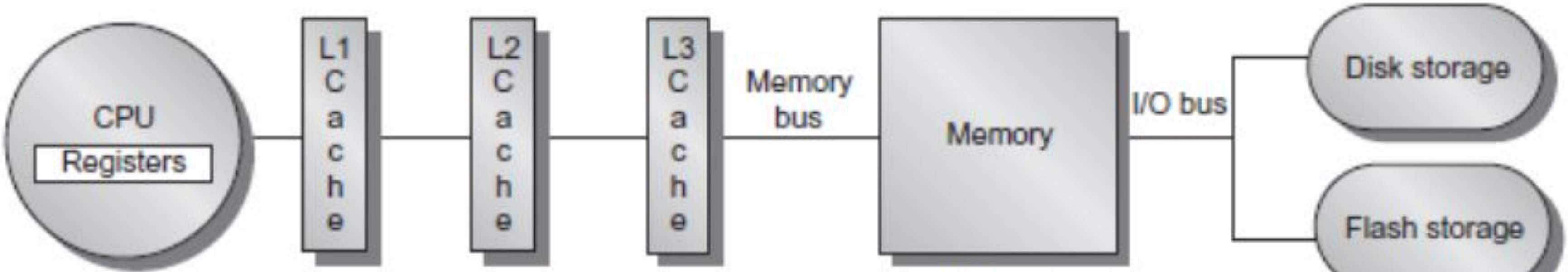
PMD(personal mobile device)

Memory Hierarchy – PC



Laptop	Size:	1000 bytes	64 KB	256 KB	4-8 MB	4–16 GB	256 GB-1 TB
	Speed:	300 ps	1 ns	3–10 ns	10 – 20 ns	50 – 100 ns	50-100 uS
Desktop	Size:	2000 bytes	64 KB	256 KB	8-32 MB	8–64 GB	256 GB-2 TB
	Speed:	300 ps	1 ns	3–10 ns	10 – 20 ns	50 – 100 ns	50-100 uS

Memory Hierarchy – Server



CPU

Registers

L1 Cache

L2 Cache

L3 Cache

Memory bus

Memory

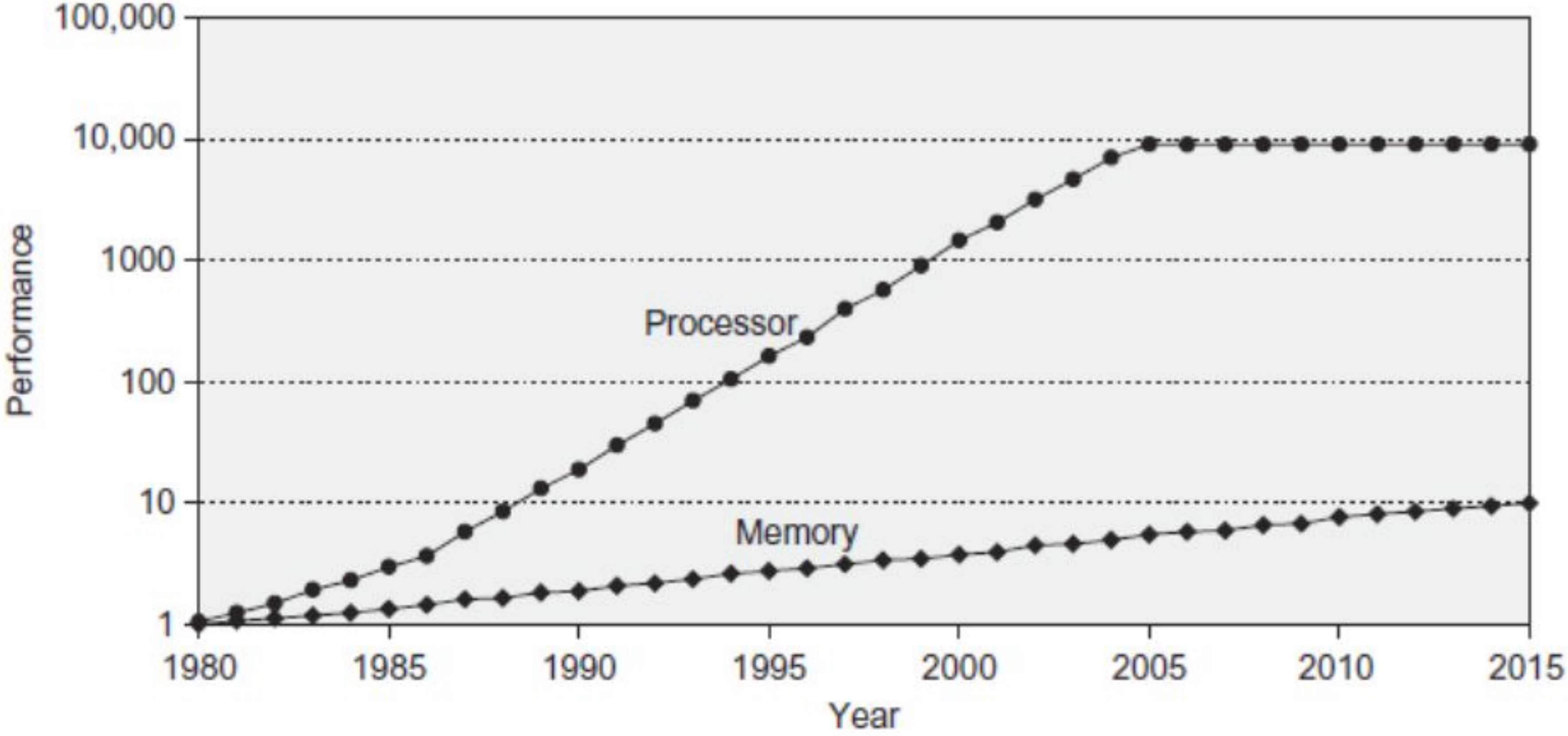
I/O bus

Disk storage

Flash storage

	Register reference	Level 1 Cache reference	Level 2 Cache reference	Level 3 Cache reference	Memory reference	Disk memory reference	Flash memory reference
Size:	4000 bytes	64 KB	256 KB	16-64 MB	32–256 GB	16–64 TB	1-16 TB
Speed:	200 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	5–10 ms	100-200 us

Memory Performance Gap



Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multi-core processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 can generate two references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion 64-bit data references/second +
      - 12.8 billion 128-bit instruction references/second
      - = 409.6 GB/s!
    - DRAM bandwidth is only 8% of this (34.1 GB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip

Performance and Power

- High-end microprocessors have >10 MB on-chip cache
  - Consumes large amount of area and power budget

Memory Hierarchy Basics

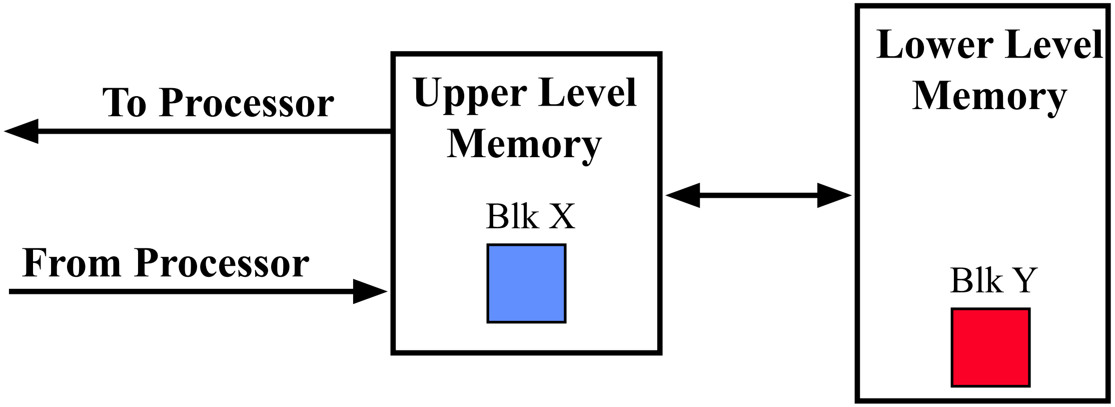
- When a word is not found in the cache, a *miss* occurs:
  - Fetch word from lower level in hierarchy, requiring a higher latency reference
  - Lower level may be another cache or the main memory
  - Also fetch the other words contained within the *block*
    - Takes advantage of spatial locality
  - Place block into cache in any location within its *set*, determined by address
    - block address MOD number of sets in cache

Memory Hierarchy Basics

- Hit: data appears in some block in the upper level (eg: Block X)
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of

    RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieved from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level +

    Time to deliver the block to the processor
- Hit Time << Miss Penalty (e.g. 500 instructions)



Memory Hierarchy Basics

- *Hit rate*: fraction found in that level
  - So high that usually talk about *Miss rate*
- Average memory-access time

  = Hit time + Miss rate x Miss penalty (ns or clocks)
- *Miss penalty*: time to replace a block from lower level, including time to replace in CPU
  - *access time*: time to lower level

    = f(latency to lower level)
  - *transfer time*: time to transfer block

    = f(BW between upper & lower levels, block size)

Memory Hierarchy Basics

- *n* sets => *n-way set associative*
  - *Direct-mapped cache* => one block per set (one way)
  - *Fully associative* => one set

- Writing to cache: two strategies
  - *Write-through*
    - Immediately update lower levels of hierarchy
  - *Write-back*
    - Only update lower levels of hierarchy when an updated block is replaced
  - Both strategies use *write buffer* to make writes asynchronous

Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that results in a miss

- Causes of misses
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

Memory Hierarchy Basics

$$\frac{\mathit{Misses}}{\mathit{Instruction}} = \frac{\mathit{Miss\ rate} \times \mathit{Memory\ accesses}}{\mathit{Instruction\ count}} = \mathit{Miss\ rate} \times \frac{\mathit{Memory\ accesses}}{\mathit{Instruction}}$$

$$\mathit{Average\ memory\ access\ time} = \mathit{Hit\ time} + \mathit{Miss\ rate} \times \mathit{Miss\ penalty}$$

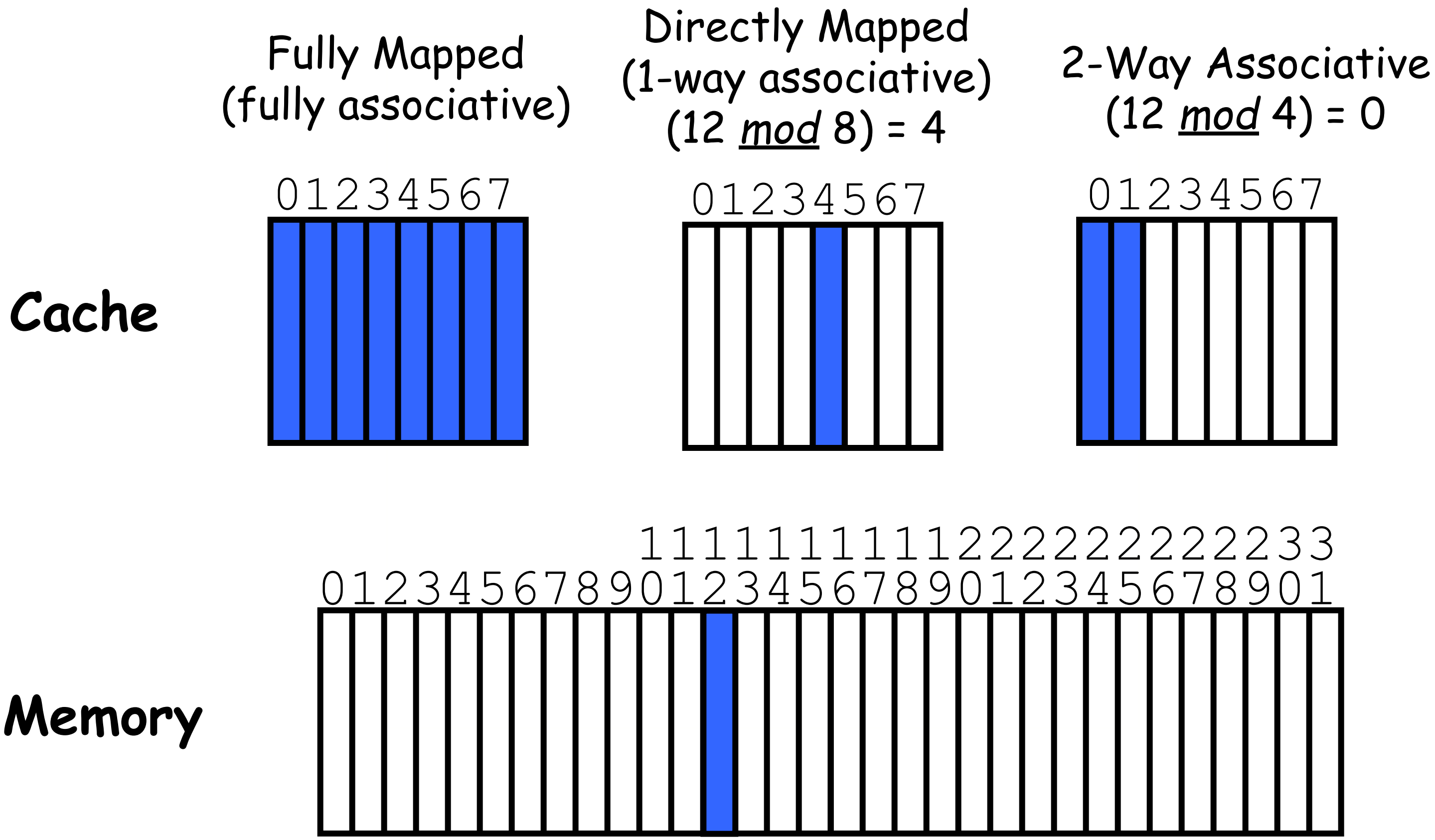
- Speculative and multithreaded processors may execute other instructions during a miss
  - Reduces performance impact of misses

Traditional Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level?
  - Block placement
    - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level?
  - Block identification
    - Tag/Block
- Q3: Which block should be replaced on a miss?
  - Block replacement
    - Random, LRU, FIFO
      - LRU (Least Recently Used), FIFO (First In-First Out)
- Q4: What happens on a write?
  - Write strategy
    - Write Back or Write Through (with Write Buffer)

Q1: Where can a block be placed in the upper level?

- Block 12 placed in an 8-block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = (Block Number) *Modulo* (Number Sets)

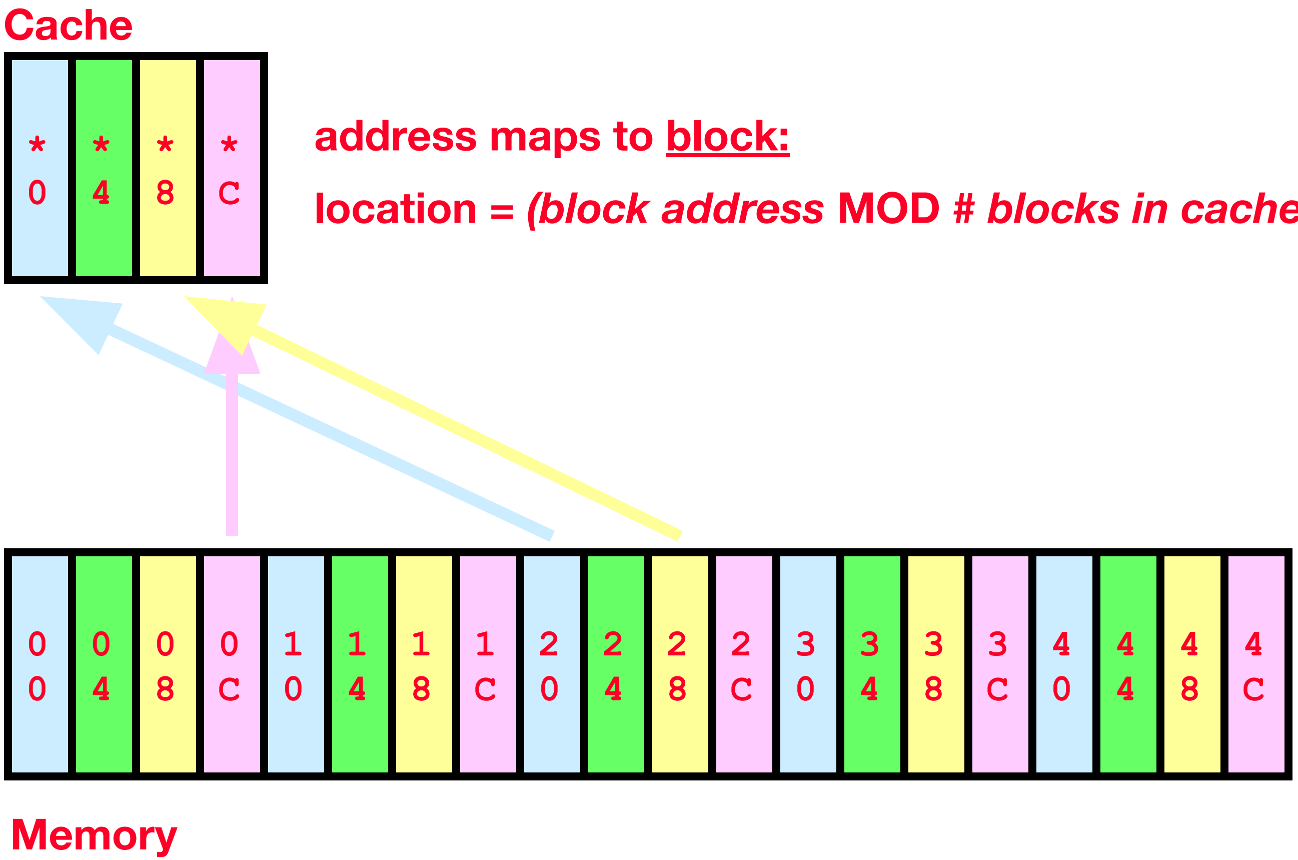


Fully Mapped
(fully associative)

Directly Mapped
(1-way associative)
(12 *mod* 8) = 4

2-Way Associative
(12 *mod* 4) = 0

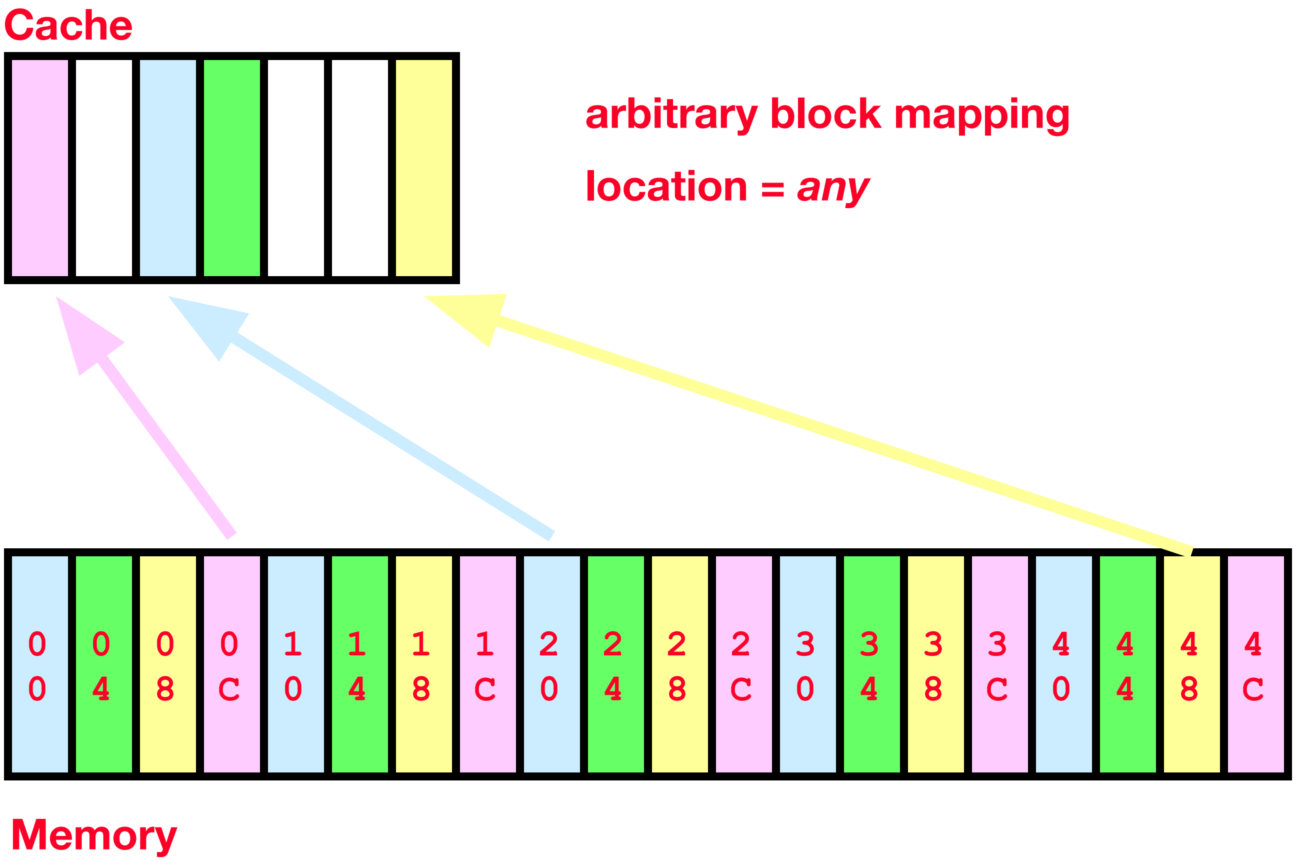
Direct Mapped Block Placement



address maps to <u>block:</u>

location = *(block address* MOD # *blocks in cache)*

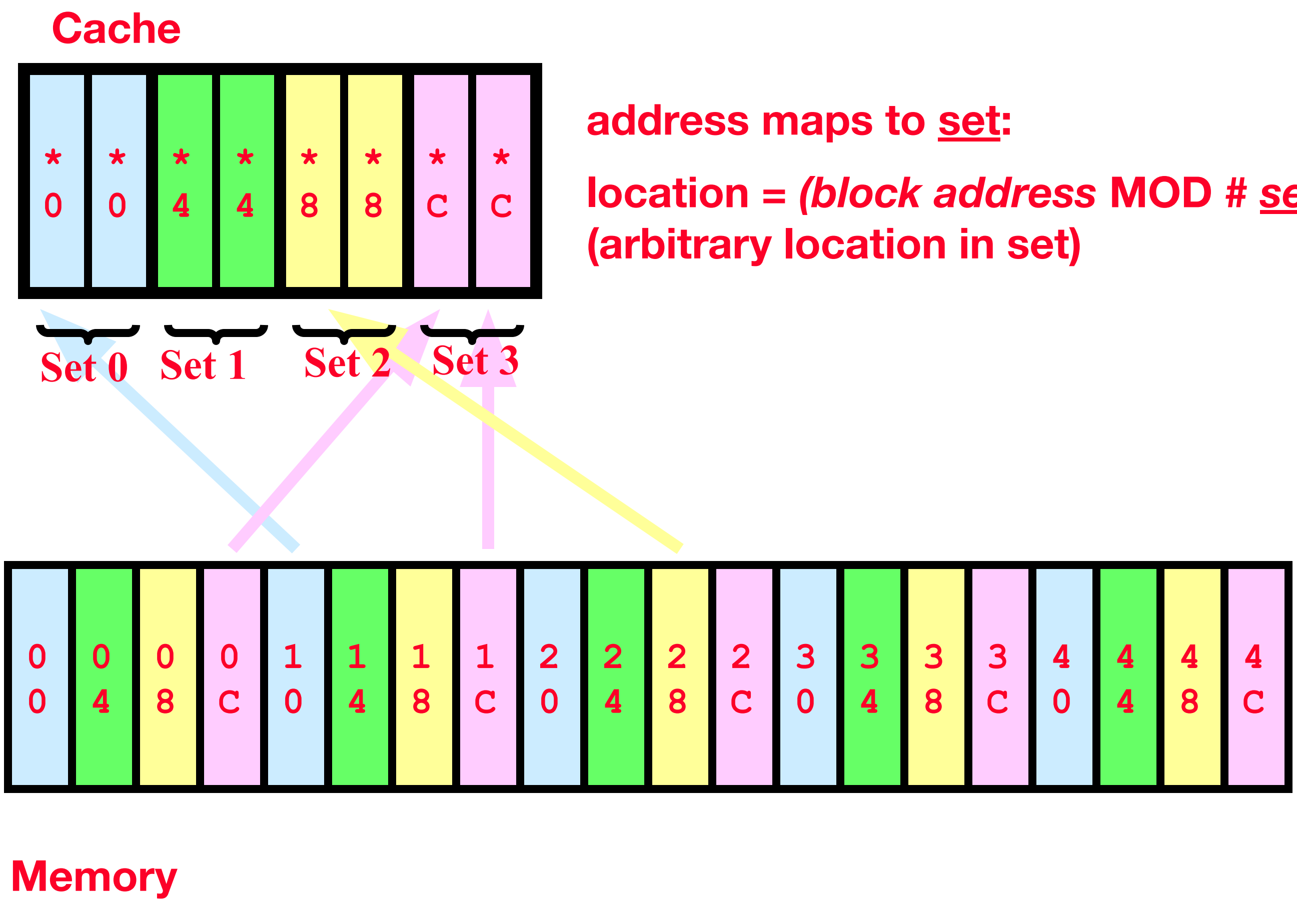
Fully Associative Block Placement



arbitrary block mapping

location = *any*

Set-Associative Block Placement

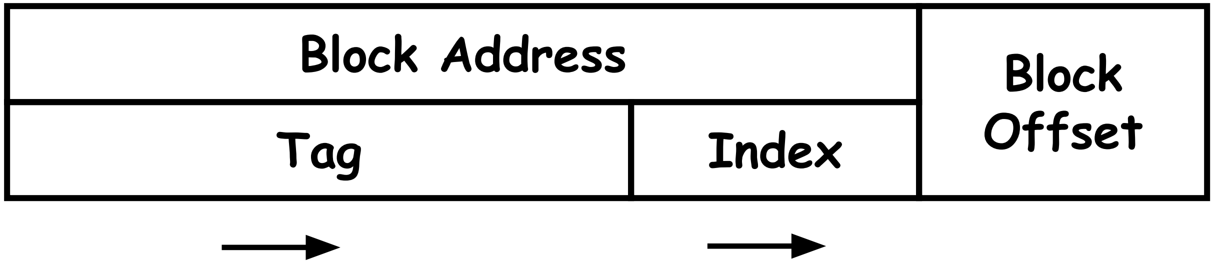


address maps to set:

location = (*block address* MOD # *sets in cache*)
(arbitrary location in set)

Q2: How is a block found if it is in the upper level?

- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag



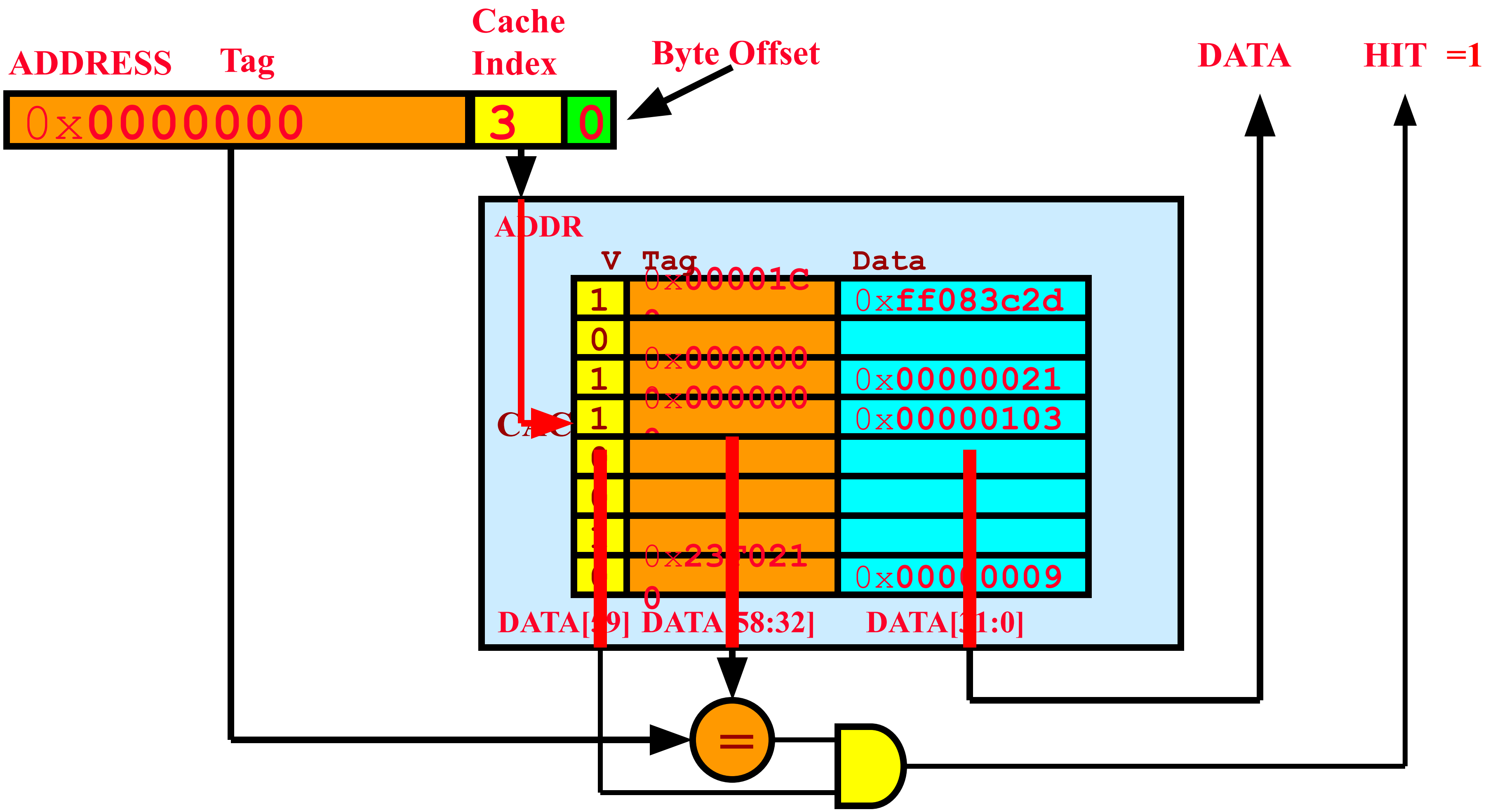
Block Address

Tag

Index

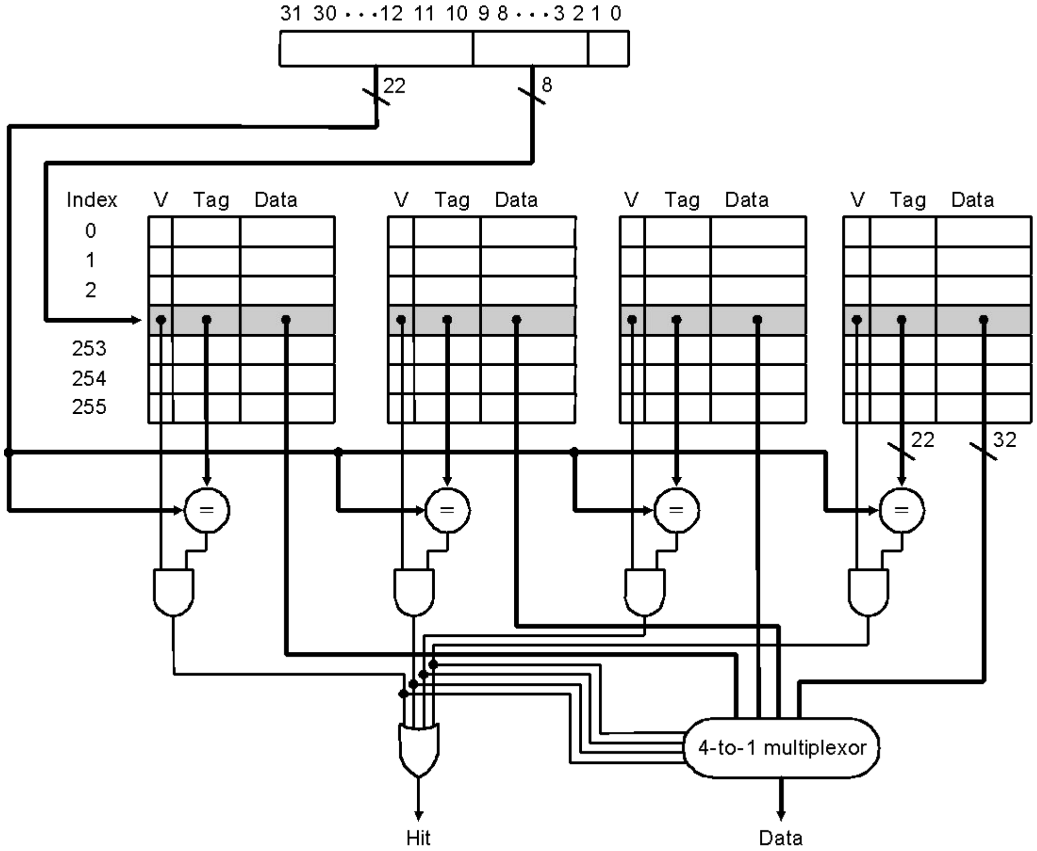
Block Offset

Direct-Mapped Cache Design



Set Associative Cache Design

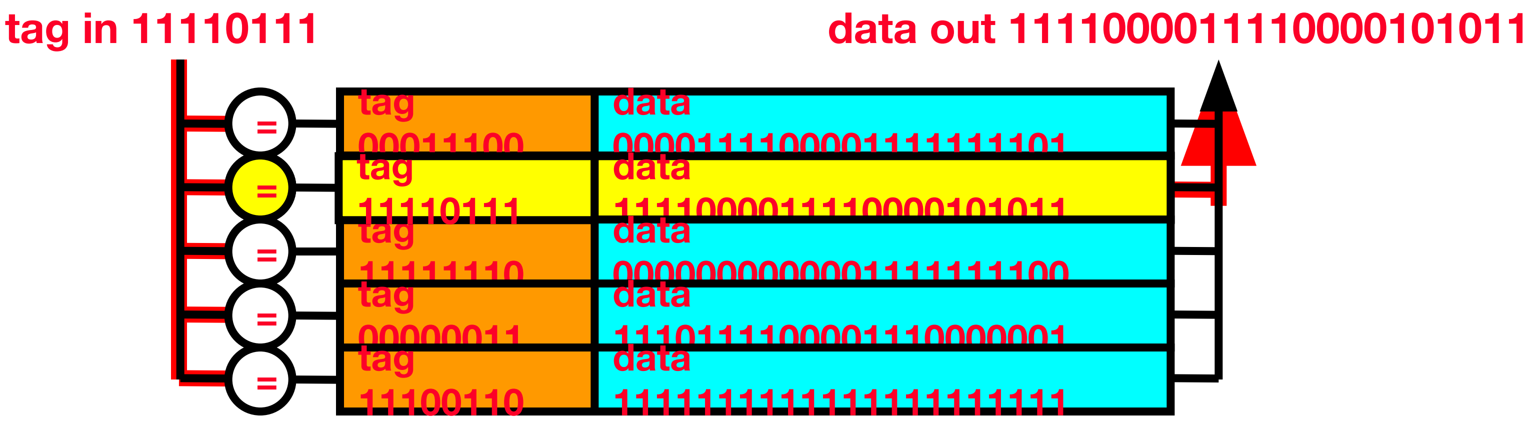
- Key idea:
  - Divide cache into sets
  - Allow block anywhere in a set
- Advantages:
  - Better hit rate
- Disadvantage:
  - More tag bits
  - More hardware
  - Higher access time



A Four-Way Set-Associative Cache

Fully Associative Cache Design

- Key idea: set size of one block
  - 1 comparator required for each block
  - No address decoding
  - Practical only for small caches due to hardware demands


tag in 11110111
data out 1111000011110000101011
=
=
=
=
=
tag 00011100
data 0000111100001111111101
tag 11110111
data 1111000011110000101011
tag 11111110
data 0000000000001111111100
tag 00000011
data 1110111100001110000001
tag 11100110
data 1111111111111111111111

Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q3: Which block should be replaced on a miss?

After a cache read miss, if there are no empty cache blocks, which block should be removed from the cache?

THE LEAST RECENTLY USED (LRU) BLOCK? APPEALING, BUT HARD TO IMPLEMENT FOR HIGH ASSOCIATIVITY

A RANDOMLY CHOSEN BLOCK? EASY TO IMPLEMENT, HOW WELL DOES IT WORK?

Miss Rate for 2-way Set Associative Cache

Size	Random	LRU
16 KB	5.7%	5.2%
64 KB	2.0%	1.9%
256 KB	1.17%	1.15%

ALSO, TRY OTHER LRU APPROX.

Q4: What happens on a write?

- Write-through: all writes update cache and underlying memory/cache
  - Can always discard cached data - most up-to-date data is in memory
  - Cache control bit: only a *valid* bit
- Write-back: all writes simply update cache
  - Can't just discard cached data - may have to write it back to memory
  - Cache control bits: both *valid* and *dirty* bits
- Other Advantages:
  - Write-through:
    - memory (or other processors) always have latest data
    - Simpler management of cache
  - Write-back:
    - much lower bandwidth, since data often overwritten multiple times
    - Better tolerance to long-latency memory?

Write Policy: What happens on write-miss?

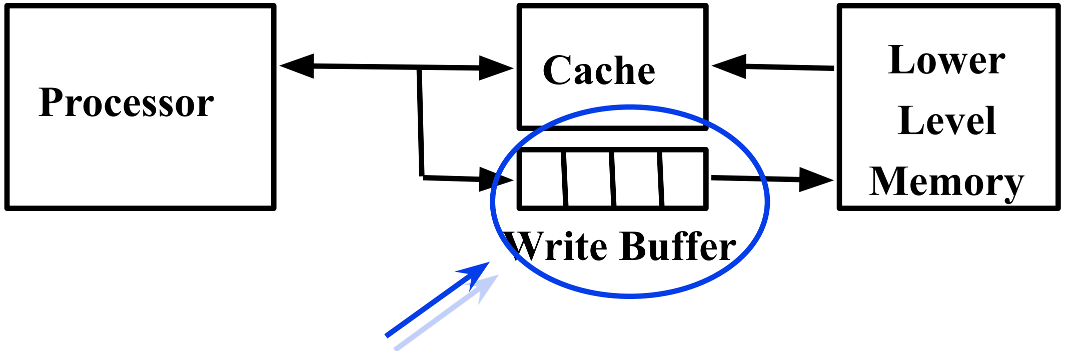
- Write allocate: allocate new cache line in cache
  - Usually means that you have to do a "read miss" to fill in rest of the cache-line!
  - Alternative: per/word valid bits
- Write non-allocate (or "write-around"):
  - Simply send write data through to underlying memory/cache - don't allocate new cache line!

Q4: What happens on a write?

	Write-Through	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data only to the cache Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

ADDITIONAL OPTION (ON MISS)-- LET WRITES TO AN UN-CACHED ADDRESS; ALLOCATE A NEW CACHE LINE ("WRITE-ALLOCATE").

Write Buffers for Write-Through Caches



Holds data awaiting write-through to lower level memory

Q. WHY A WRITE BUFFER

A. SO CPU DOESN'T STALL

Q. WHY A BUFFER, WHY NOT JUST ONE REGISTER

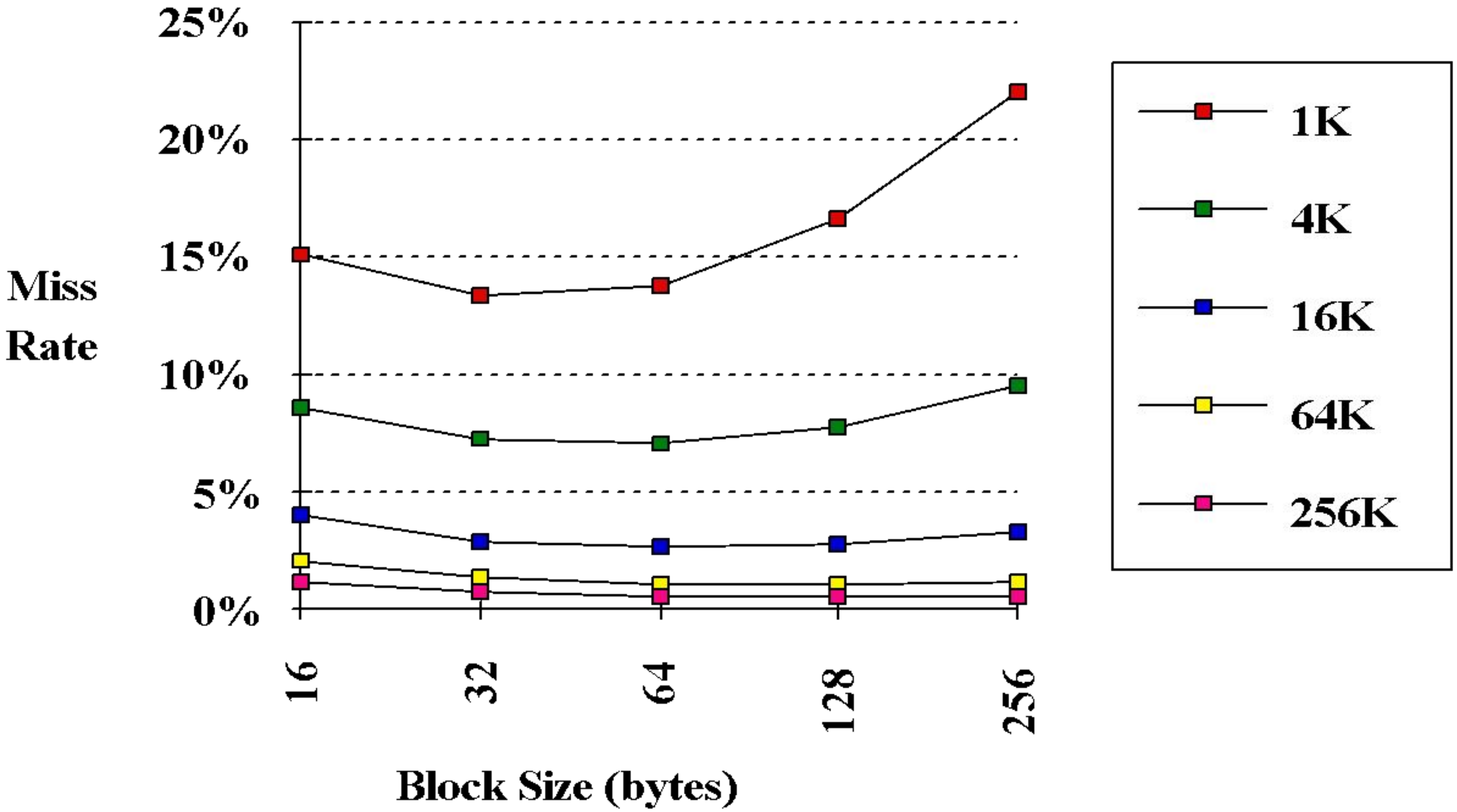
A. BURSTS OF WRITES ARE COMMON.

Q. ARE READ AFTER WRITE (RAW) HAZARDS AN ISSUE FOR WRITE BUFFER?

A. YES: DRAIN BUFFER BEFORE NEXT READ, OR SEND READ 1ST AFTER CHECK WRITE BUFFERS.

Reducing Cache Misses: 1. Larger Block Size

- Using the principle of locality. The larger the block, the greater the chance parts of it will be used again.



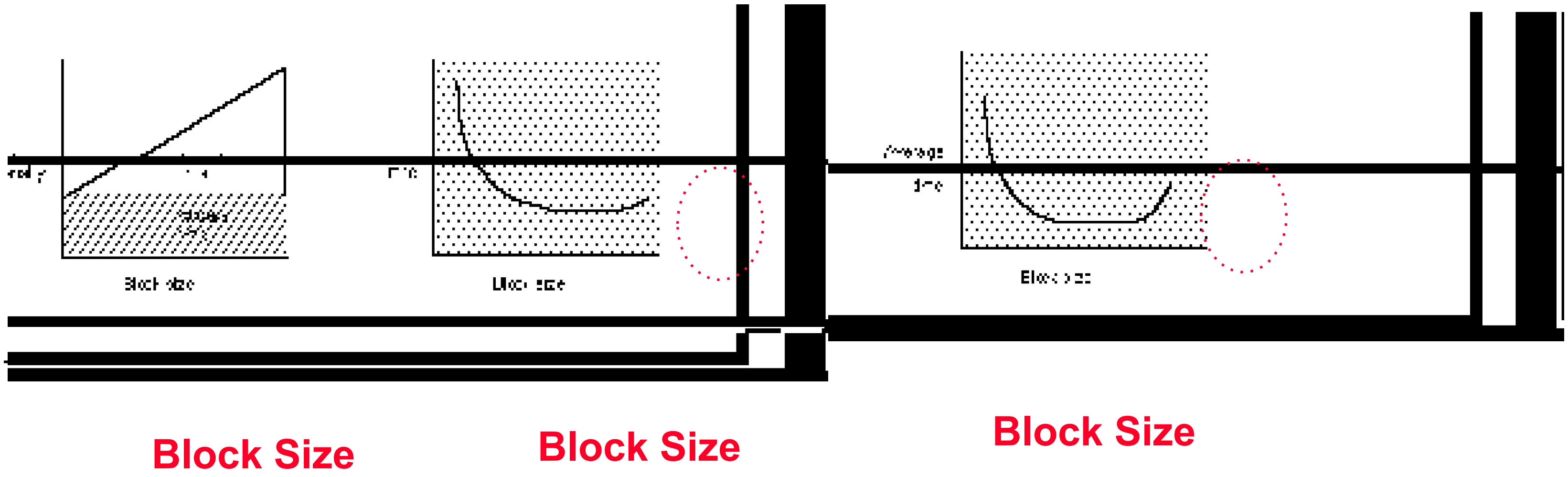
Increasing Block Size

- One way to reduce the miss rate is to increase the block size
  - Take advantage of spatial locality
  - Decreases compulsory misses
- However, larger blocks have disadvantages
  - May increase the miss penalty (need to get more data)
  - May increase hit time (need to read more data from cache and larger mux)
  - May increase miss rate, since conflict misses
- Increasing the block size can help, but don't overdo it.

Block Size vs. Cache Measures

- Increasing Block Size generally increases Miss Penalty and decreases Miss Rate
- As the block size increases the AMAT starts to decrease, but eventually increases

Hit Time + *Miss Penalty* x *Miss Rate* = *Avg. Memory Access Time*



Reducing Cache Misses: Higher Associativity

- Increasing associativity helps reduce conflict misses
- 2:1 Cache Rule:
  - The miss rate of a direct mapped cache of size N is about equal to the miss rate of a 2-way set associative cache of size N/2
  - For example, the miss rate of a 32 Kbyte direct mapped cache is about equal to the miss rate of a 16 Kbyte 2-way set associative cache
- Disadvantages of higher associativity
  - Need to do large number of comparisons
  - Need n-to-1 multiplexor for n-way set associative
  - Could increase hit time
  - Consume more power

AMAT vs. Associativity

Cache Size	Associativity			
(KB)	1-way	2-way	4-way	8-way
1	7.65	6.60	6.22	5.44
2	5.90	4.90	4.62	4.09
4	4.60	3.95	3.57	3.19
8	3.30	3.00	2.87	2.59
16	2.45	2.20	2.12	2.04
32	2.00	1.80	1.77	1.79
64	1.70	1.60	1.57	1.59
128	1.50	1.45	1.42	1.44

Red means A.M.A.T. not improved by more associativity

Does not take into account effect of slower clock on rest of program

Cache performance

- Miss-oriented Approach to Memory Access:

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemMisses}{Inst} \times MissPenalty \right) \times CycleTime$$

  - $CPI_{Execution}$ includes ALU and Memory instructions

- Separating out Memory component entirely
  - AMAT = Average Memory Access Time
  - CPI_{ALUOps} does not include memory instructions

$$\begin{aligned} AMAT &= HitTime + MissRate \times MissPenalty \\ &= \left(HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst} \right) + \\ &\quad \left(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data} \right) \end{aligned}$$

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

Impact on Performance

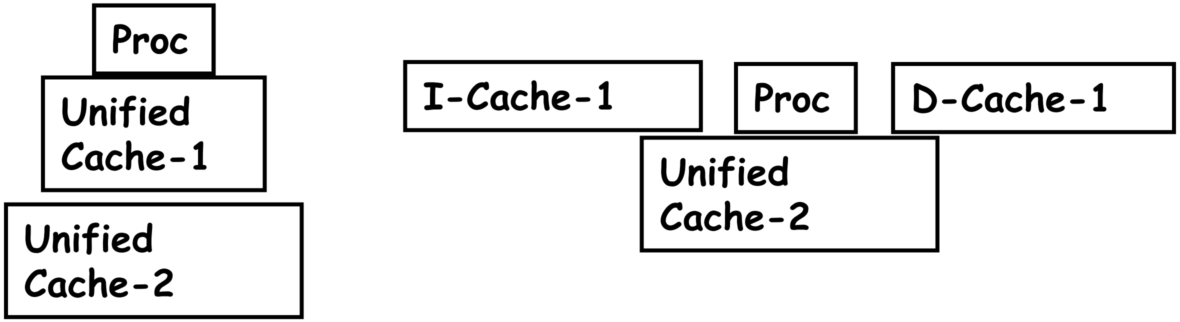
- Suppose a processor executes at
  - Clock Rate = 200 MHz (5 ns per cycle), Ideal (no misses) CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control
- Suppose that 10% of memory operations get 50 cycle miss penalty
- Suppose that 1% of instructions get same miss penalty
- CPI = ideal CPI + average stalls per instruction

1.1(cycles/ins) + [0.30 (DataMops/ins)
x 0.10 (miss/DataMop) x 50 (cycle/miss)] + [1 (InstMop/ins)
x 0.01 (miss/InstMop) x 50 (cycle/miss)]
= (1.1 + 1.5 + .5) cycle/ins = 3.1

- 58% of the time the proc is stalled waiting for memory!
  - AMAT=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=2.54

Unified vs. Split Caches

- Unified vs. Separate I&D



- Example:
  - 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
  - 32KB unified: Aggregate miss rate=1.99%
- Which is better (ignore L2 cache)?
  - Assume 33% data ops ⇒ 75% accesses from instr. (1.0/1.33)
  - hit time=1, miss time=50
  - Note that *data* hit has 1 stall for unified cache (only one port)

$$\text{AMAT}_{\text{Harvard}}=75\%\text{x}(1+0.64\%\text{x}50)+25\%\text{x}(1+6.47\%\text{x}50) = 2.05$$
$$\text{AMAT}_{\text{Unified}}=75\%\text{x}(1+1.99\%\text{x}50)+25\%\text{x}(1+1+1.99\%\text{x}50)= 2.24$$

Improve Cache Performance

- improve cache and memory access times:

Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty

Reducing each of these!

Simultaneously?

$$CPUtime = IC * (CPI_{Execution} + \frac{MemoryAccess}{Instruction} * MissRate * MissPenalty * ClockCycleTime)$$

- Improve performance by:
  - Reduce the miss rate,
  - Reduce the miss penalty, or
  - Reduce the time to hit in the cache.

Memory Hierarchy Basics

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Higher number of cache levels
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
  - Avoiding address translation in cache indexing
    - Reduces hit time

Miss Rate Reduction

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ accesses}{Instruction} \times \boxed{\textbf{Miss rate}} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ time$$

- 3 Cs: Compulsory, Capacity, Conflict

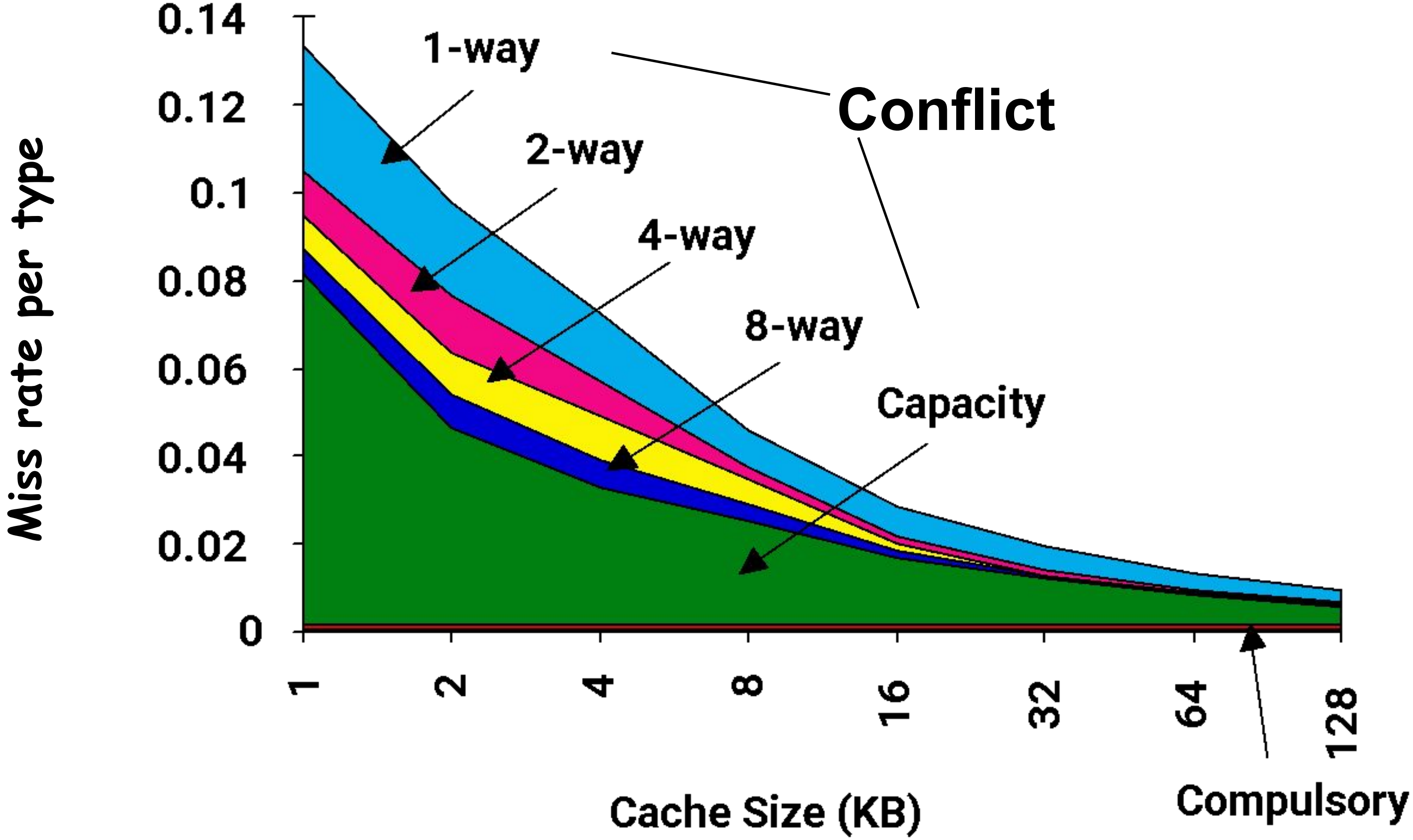
  0. Larger cache
  1. Reduce Misses via Larger Block Size
  2. Reduce Misses via Higher Associativity
  3. Reducing Misses via Victim Cache
  4. Reducing Misses via Pseudo-Associativity
  5. Reducing Misses by HW Prefetching Instr, Data
  6. Reducing Misses by SW Prefetching Data
  7. Reducing Misses by Compiler Optimizations

- Danger of concentrating on just one parameter!
- Prefetching comes in two flavors:
  - Binding prefetch: Requests load directly into register.
    - Must be correct address and register!
  - Non-Binding prefetch: Load into cache.
    - Can be incorrect. Frees HW/SW to guess!

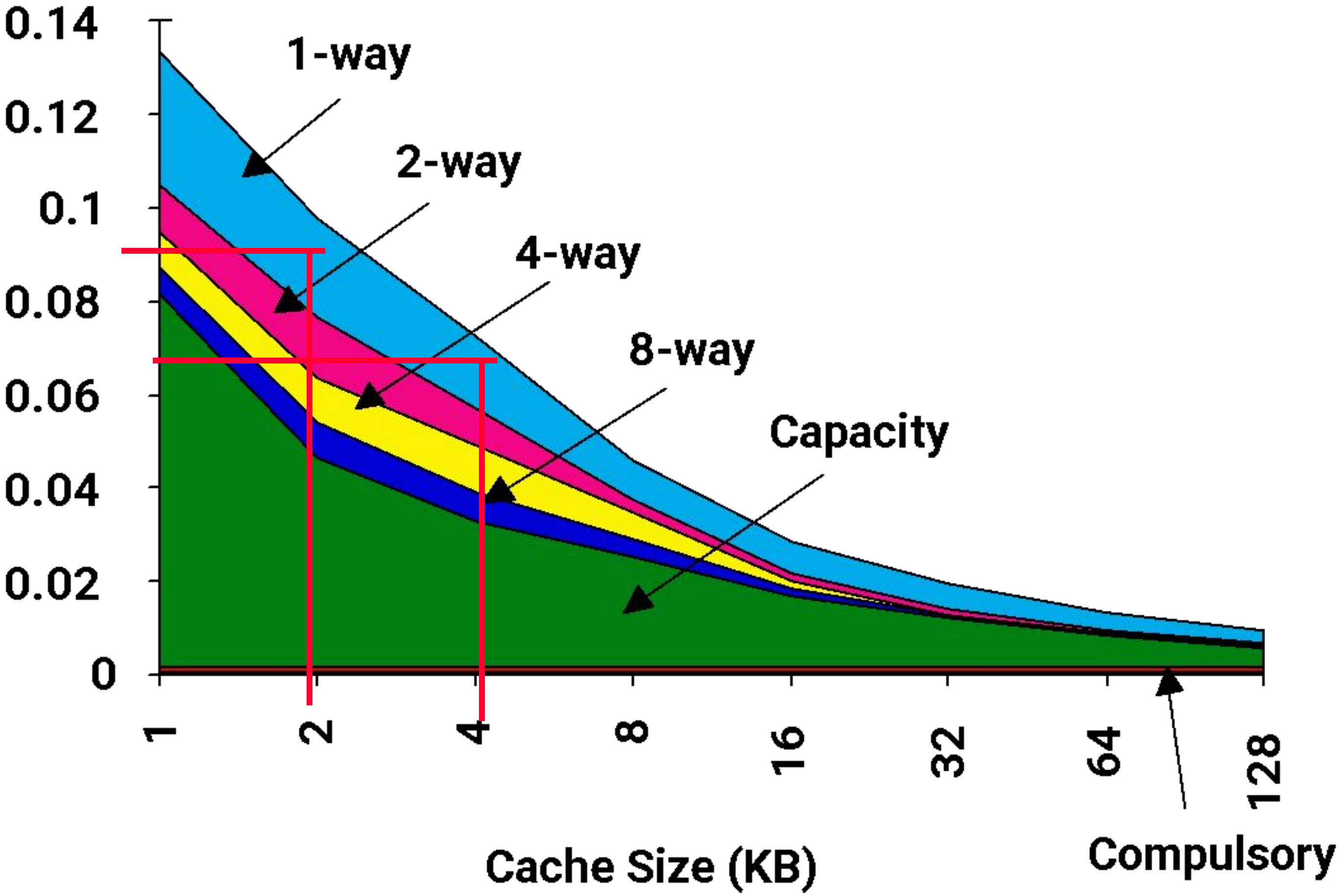
Where to misses come from?

- Classifying Misses: 3 Cs
  - Compulsory—The first access to a block is not in the cache, so the block must be brought into the cache. Also called cold start misses or first reference misses. (Misses in even an Infinite Cache)
  - Capacity—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved. (Misses in Fully Associative Size X Cache)
  - Conflict—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called collision misses or interference misses. (Misses in N-way Associative, Size X Cache)
- 4th “C”:
  - Coherence - Misses caused by cache coherence.

3Cs Absolute Miss Rate (SPEC92)



0. Cache Size



- Old rule of thumb: 2x size => 25% cut in miss rate
- What does it reduce?
- Thrashing reduction!!!

Cache Organization?

- Assume total cache size not changed:
- What happens if:

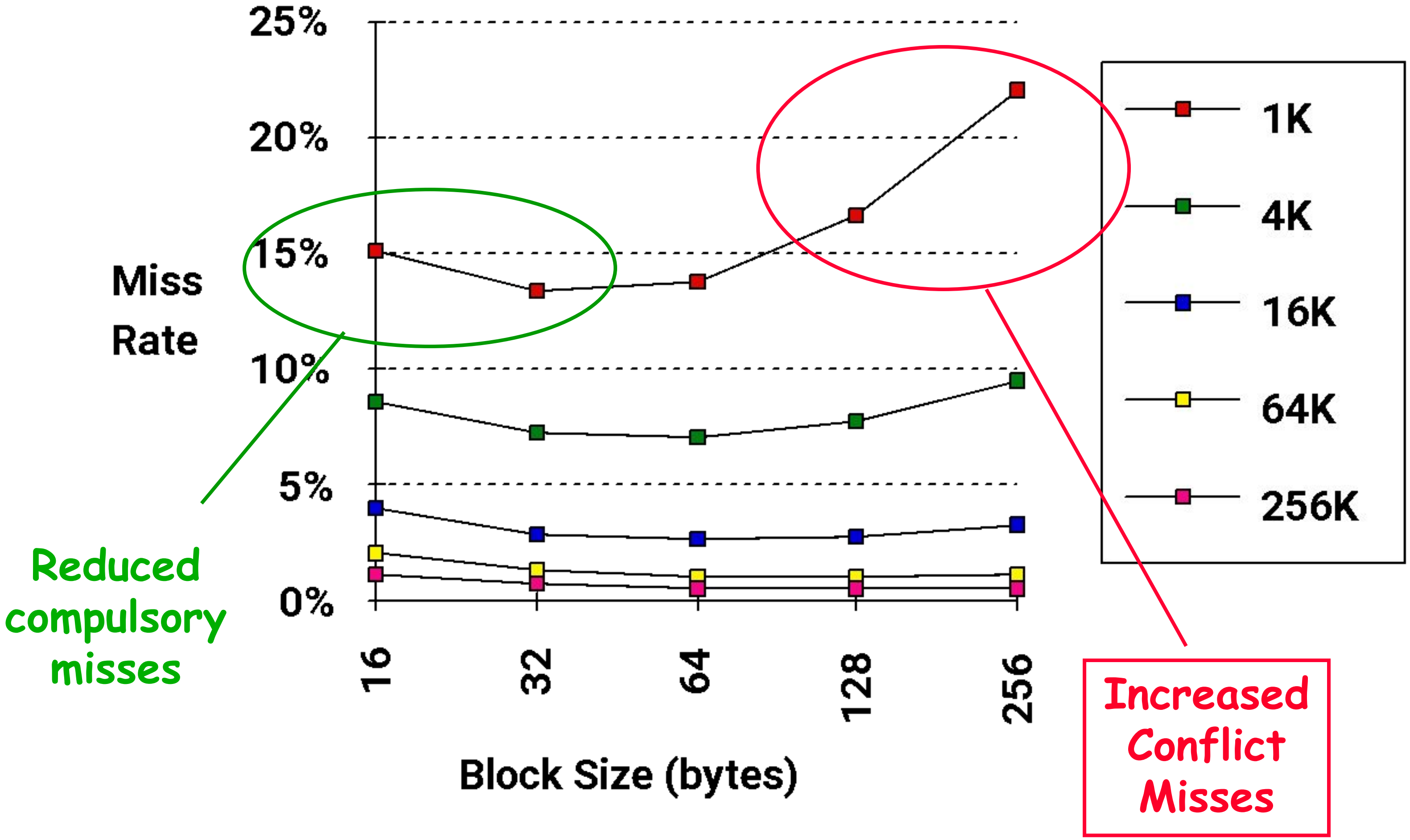
1) Change Block Size:

2) Change Associativity:

3) Change Compiler:

Which of 3Cs is obviously affected?

1. Larger Block Size (fixed size & assoc)

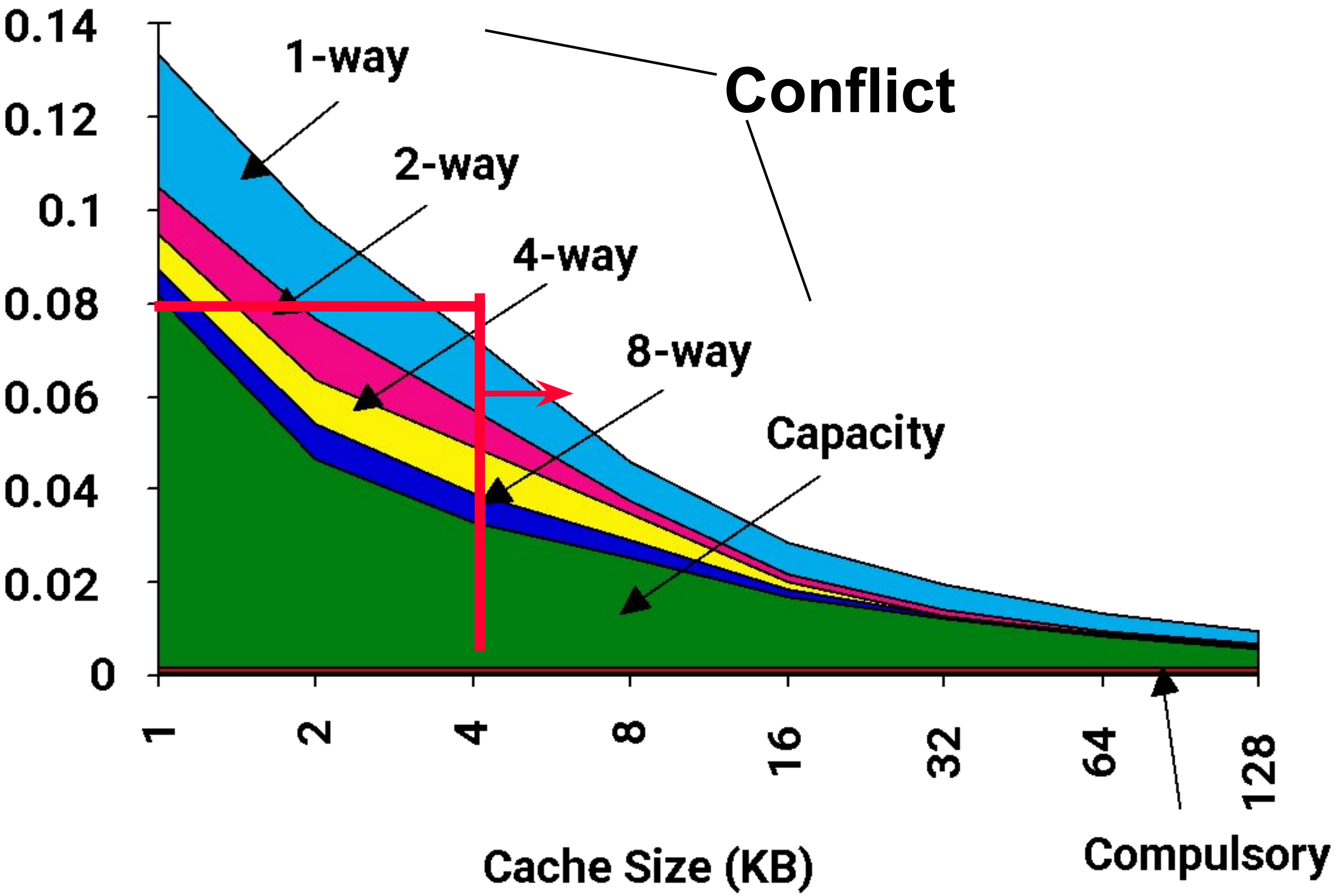


Reduced compulsory misses

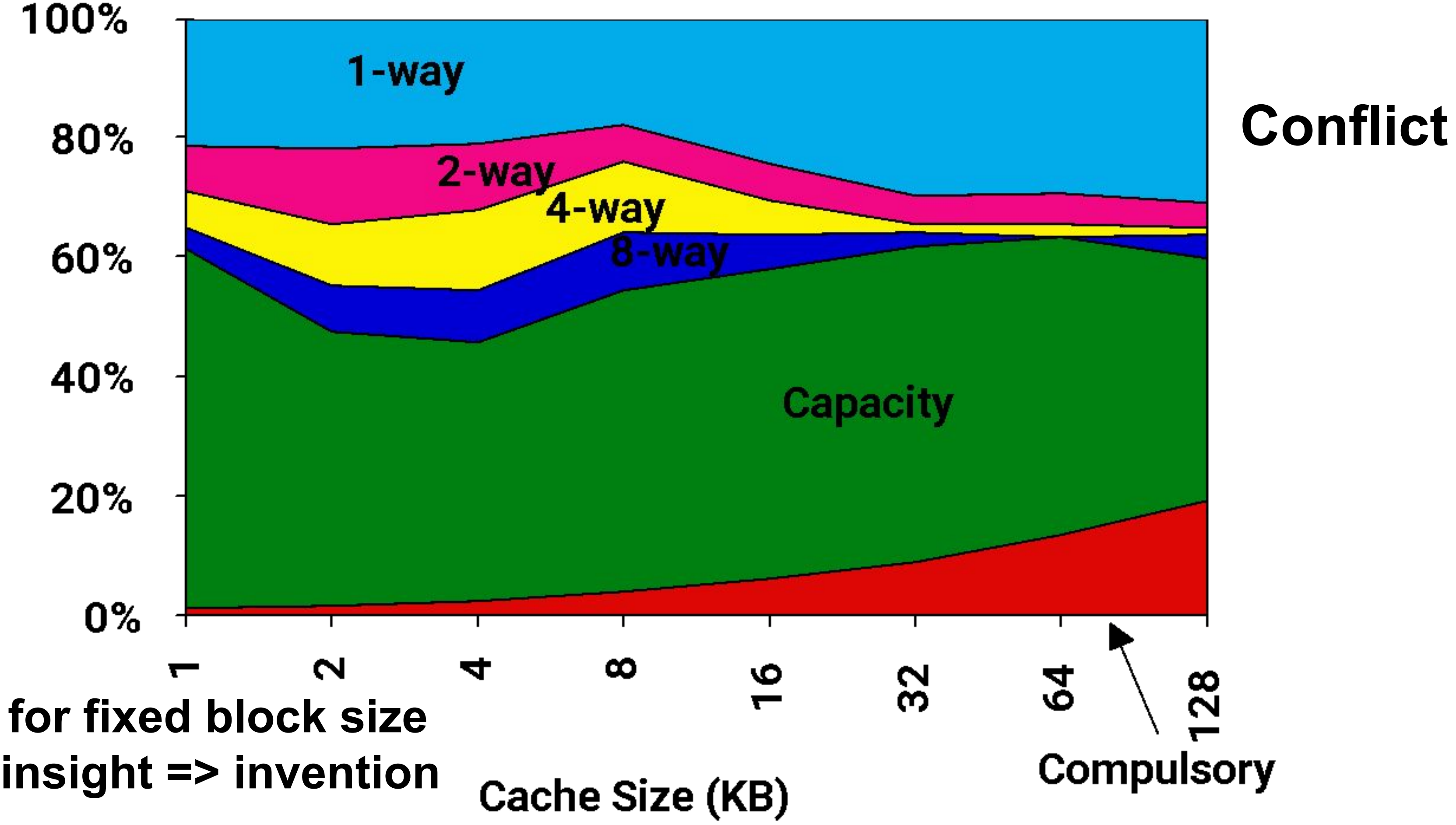
Increased Conflict Misses

What else drives up block size?

2. Higher Associativity



3Cs Relative Miss Rate



Flaws: for fixed block size
Good: insight => invention

Associativity vs. Cycle Time

- Beware: Execution time is only final measure!
- Why is cycle time tied to hit time?

- Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%, internal + 2%
  - suggested big and dumb caches

Effective cycle time of assoc
pzrbski ISCA

Example: Avg. Memory Access Time vs. Miss Rate

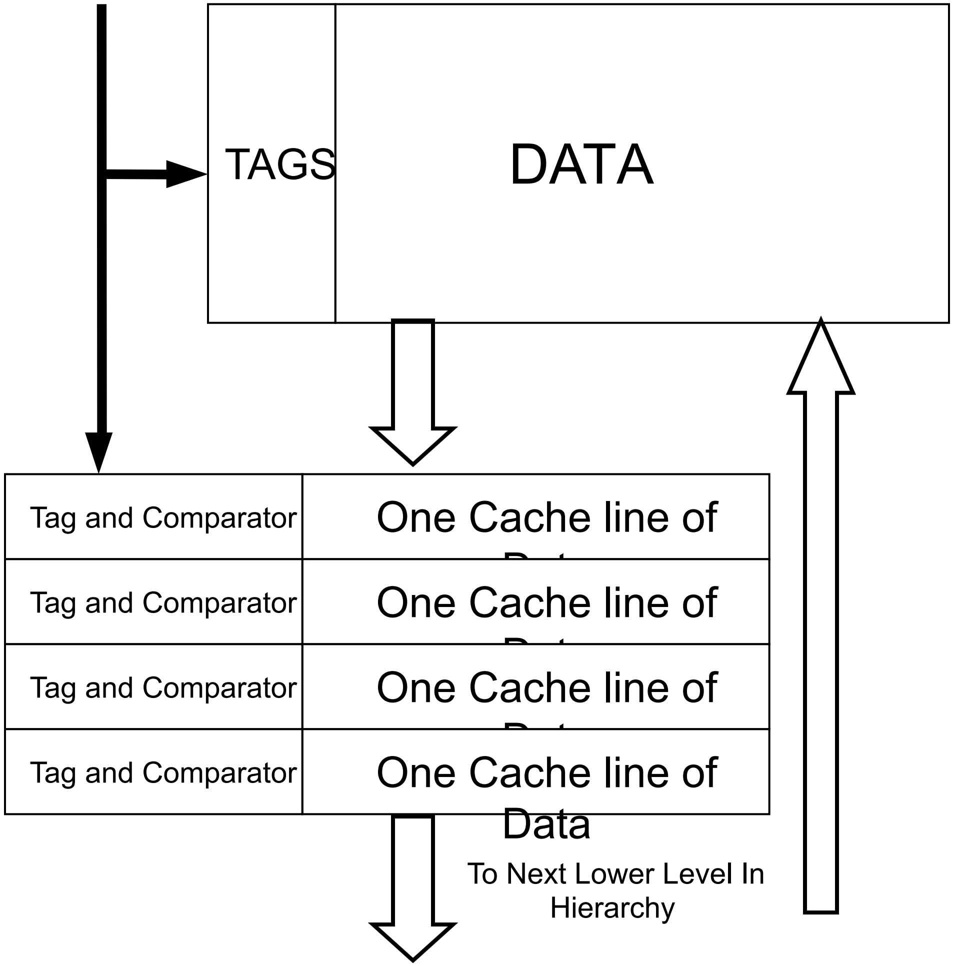
- Example: assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped

Cache Size	Associativity			
(KB)	1-way	2-way	4-way	8-way
1	2.33	2.15	2.07	2.01
2	1.98	1.86	1.76	1.68
4	1.72	1.67	1.61	1.53
8	1.46	1.48	1.47	1.43
16	1.29	1.32	1.32	1.32
32	1.20	1.24	1.25	1.27
64	1.14	1.20	1.21	1.23
128	1.10	1.17	1.18	1.20

(Red means A.M.A.T. not improved by more associativity)

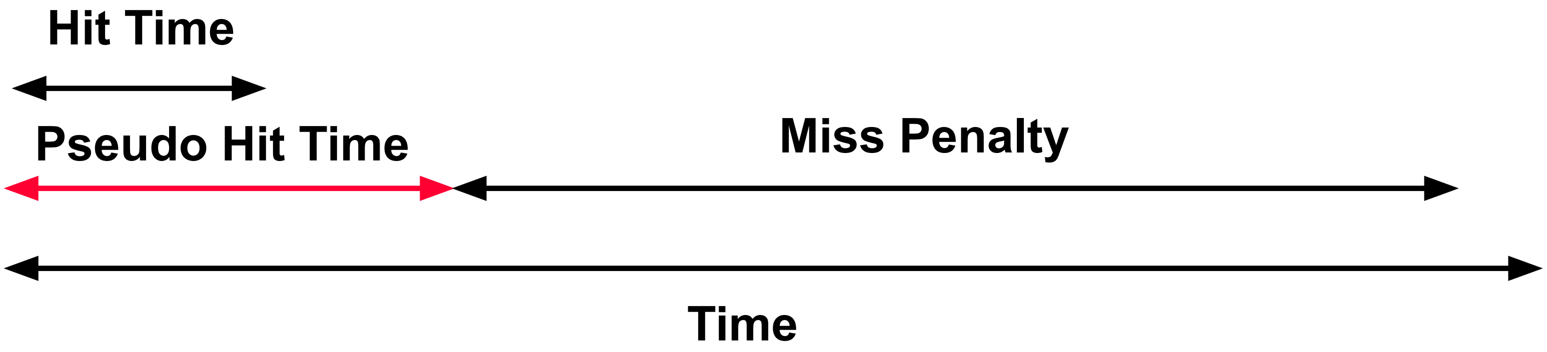
3. Victim Cache

- Fast Hit Time + Low Conflict => Victim Cache
- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines



4. Pseudo-Associativity

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit)



- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Better for caches not tied directly to processor (L2)
  - Used in MIPS R1000 L2 cache, similar in UltraSPARC

5. Hardware Prefetching of Instructions & Data

- E.g., Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in "stream buffer"
  - On miss check stream buffer
- Works with data blocks too:
  - Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches
- Prefetching relies on having extra memory bandwidth that can be used without penalty

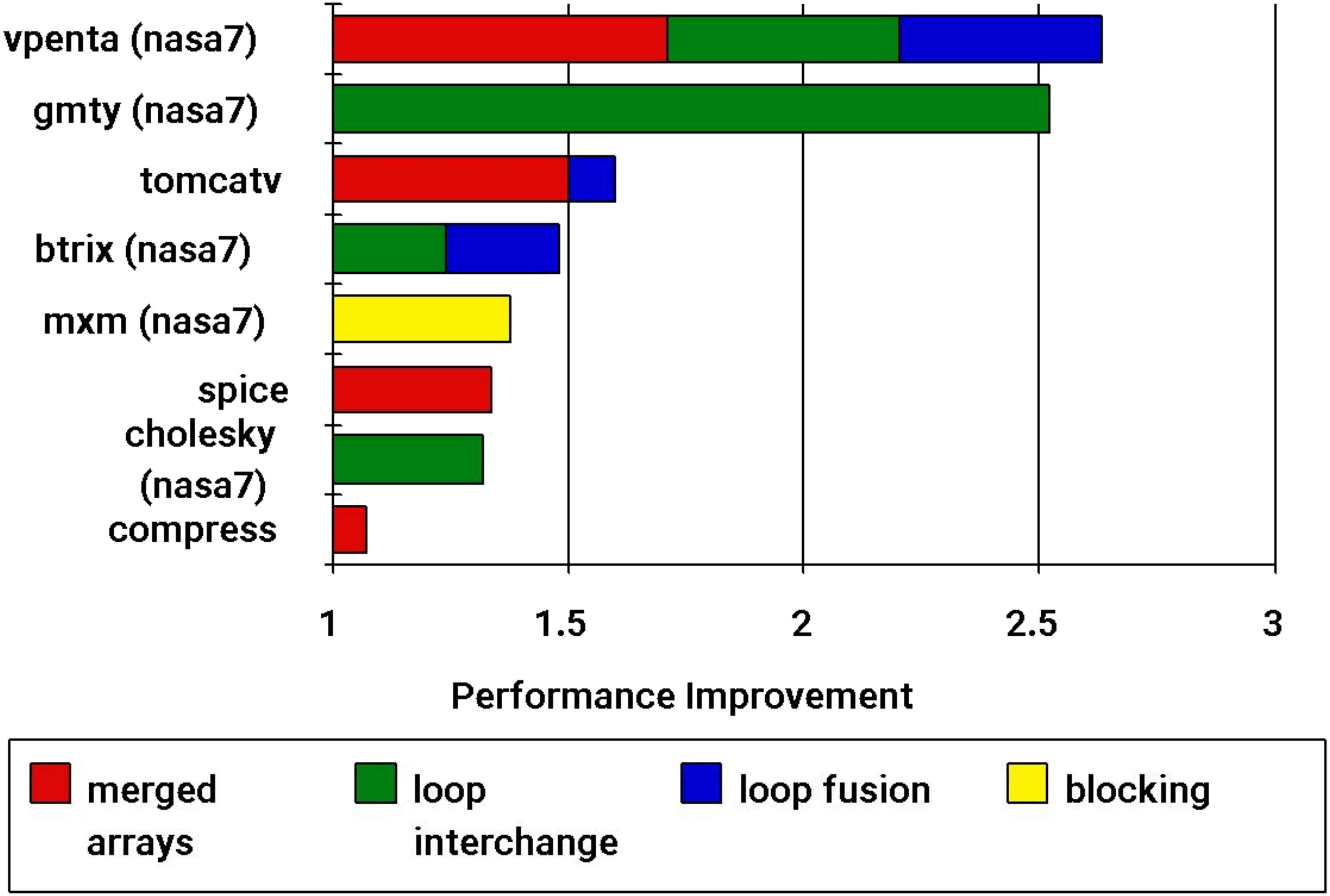
6. Software Prefetching Data

- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
- Prefetching comes in two flavors:
  - Binding prefetch: Requests load directly into register.
    - Must be correct address and register!
  - Non-Binding prefetch: Load into cache.
    - Can be incorrect. Faults?
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

7. Compiler Optimizations

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts(using tools they developed)
- Data
  - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
  - *Loop Interchange*: change nesting of loops to access data in order stored in memory
  - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
  - *Blocking*: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows

Summary of Compiler Optimizations to Reduce Cache Misses (by hand)



Improving Cache Performance

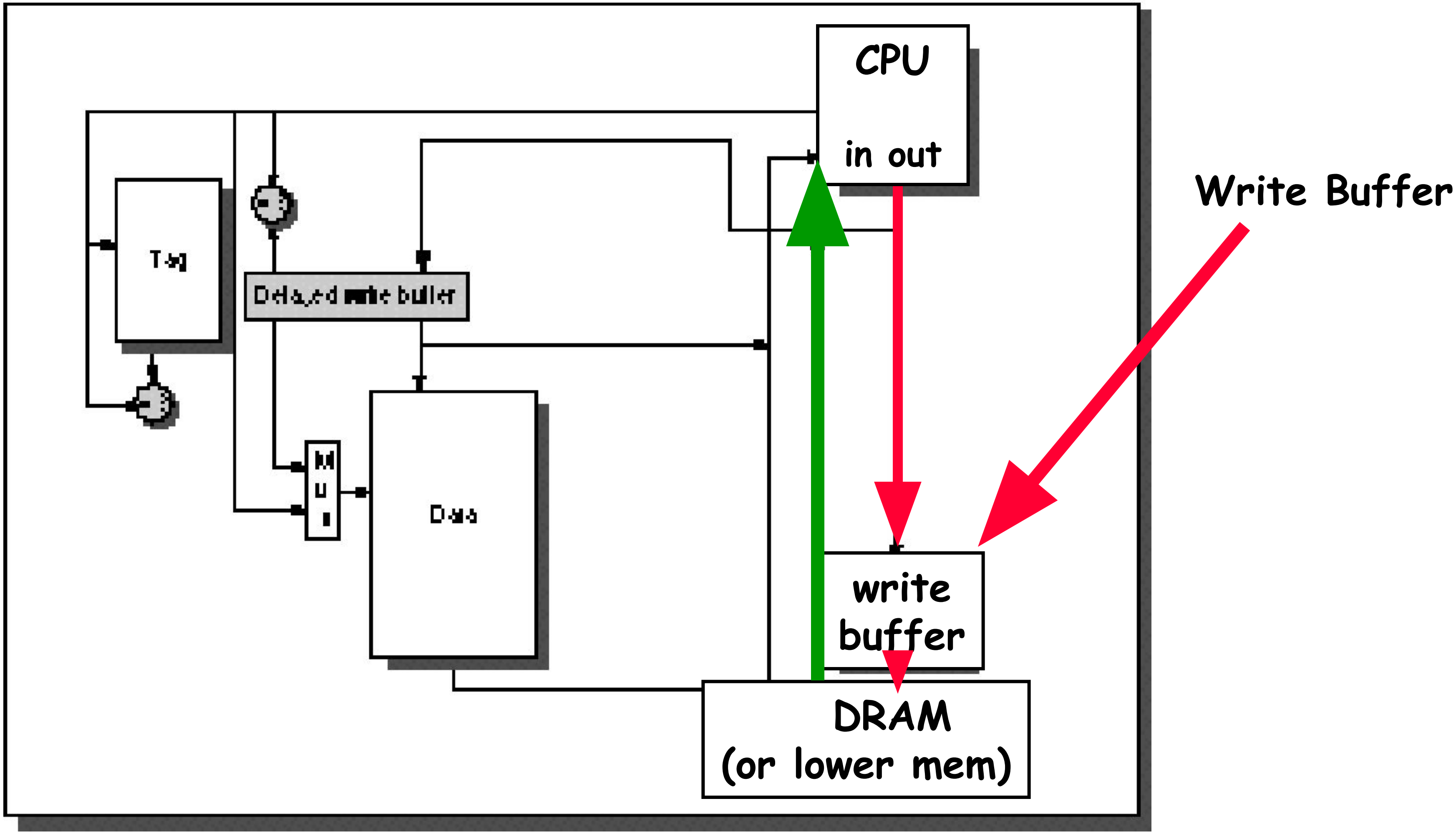
1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

Reducing Miss Penalty

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ accesses}{Instruction} \times \mathbf{Miss\ rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ time$$

- Four techniques
  - Read priority over write on miss
  - Early Restart and Critical Word First on miss
  - Non-blocking Caches (Hit under Miss, Miss under Miss)
  - Second Level Cache
- Can be applied recursively to Multilevel Caches
  - Danger is that time to DRAM will grow with multiple levels in between
  - First attempts at L2 caches can make things worse, since increased worst case is worse
- Out-of-order CPU can hide L1 data cache miss (3–5 clocks), but stall on L2 miss (40–100 clocks)?

1. Read Priority over Write on Miss



1. Read Priority over Write on Miss

- Write-through w/ write buffers => RAW conflicts with main memory reads on cache misses
  - If simply wait for write buffer to empty, might increase read miss penalty (old MIPS 1000 by 50%)
  - Check write buffer contents before read;
    if no conflicts, let the memory access continue
- Write-back want buffer to hold displaced blocks
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead copy the dirty block to a write buffer, then do the read, and then do the write
  - CPU stall less since restarts as soon as do read

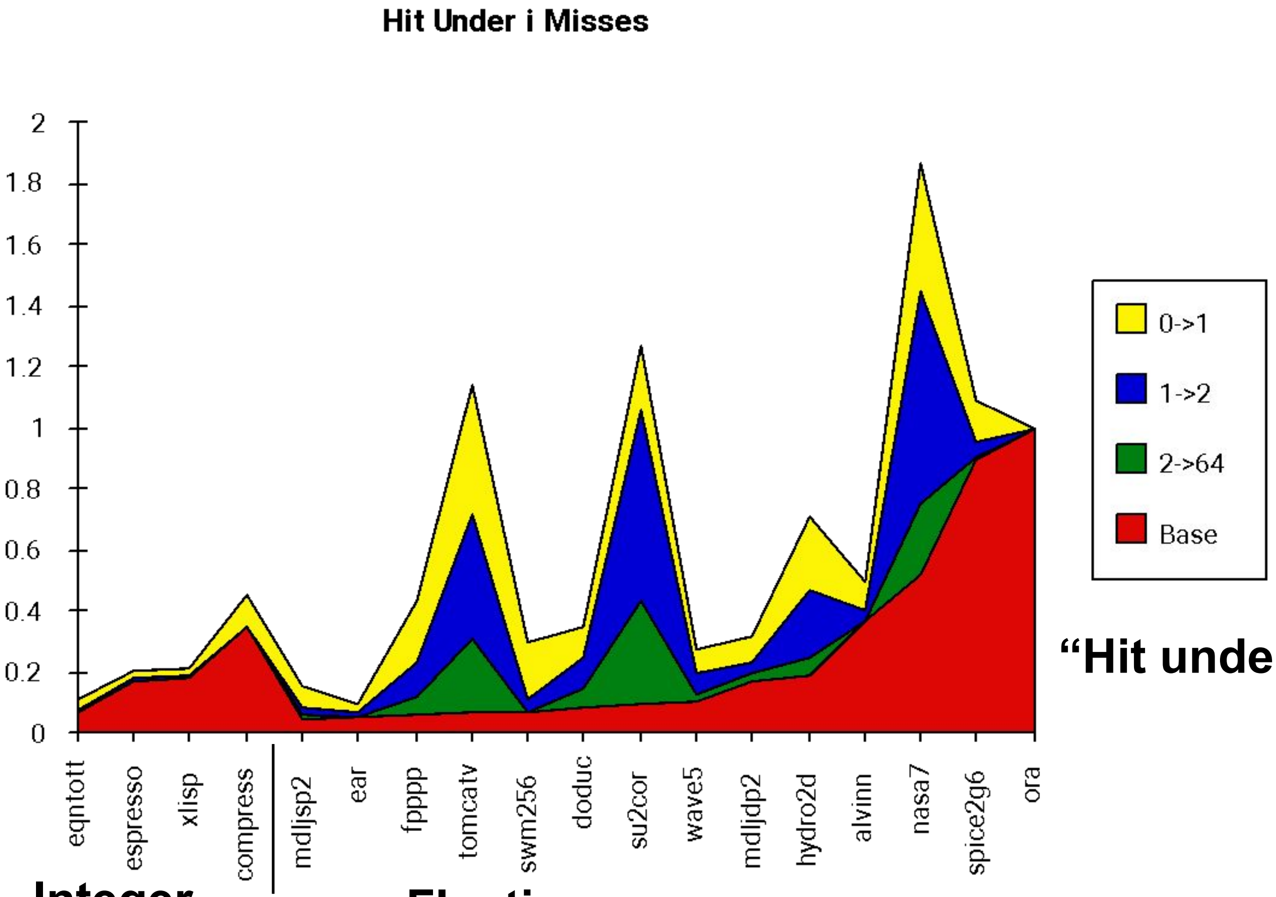
2. Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU
  - Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called wrapped fetch and requested word first
- Generally useful only in large blocks,
- Spatial locality => tend to want next sequential word, so not clear if benefit by early restart

3. Non-blocking Caches

- Non-blocking cache or lockup-free cache allow data cache to continue to supply cache hits during a miss
  - requires F/E bits on registers or out-of-order execution
  - requires multi-bank memories
- "hit under miss" reduces the effective miss penalty by working during miss vs.. ignoring CPU requests
- "hit under multiple miss" or "miss under miss" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires multiples memory banks (otherwise cannot support)
  - Pentium Pro allows 4 outstanding memory misses

Value of Hit Under Miss for SPEC



0->1
1->2
2->64
Base

"Hit under n Misses"

Integer

Floating Point

- FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

4. Add a Second-level Cache

- L2 Equations

  AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$

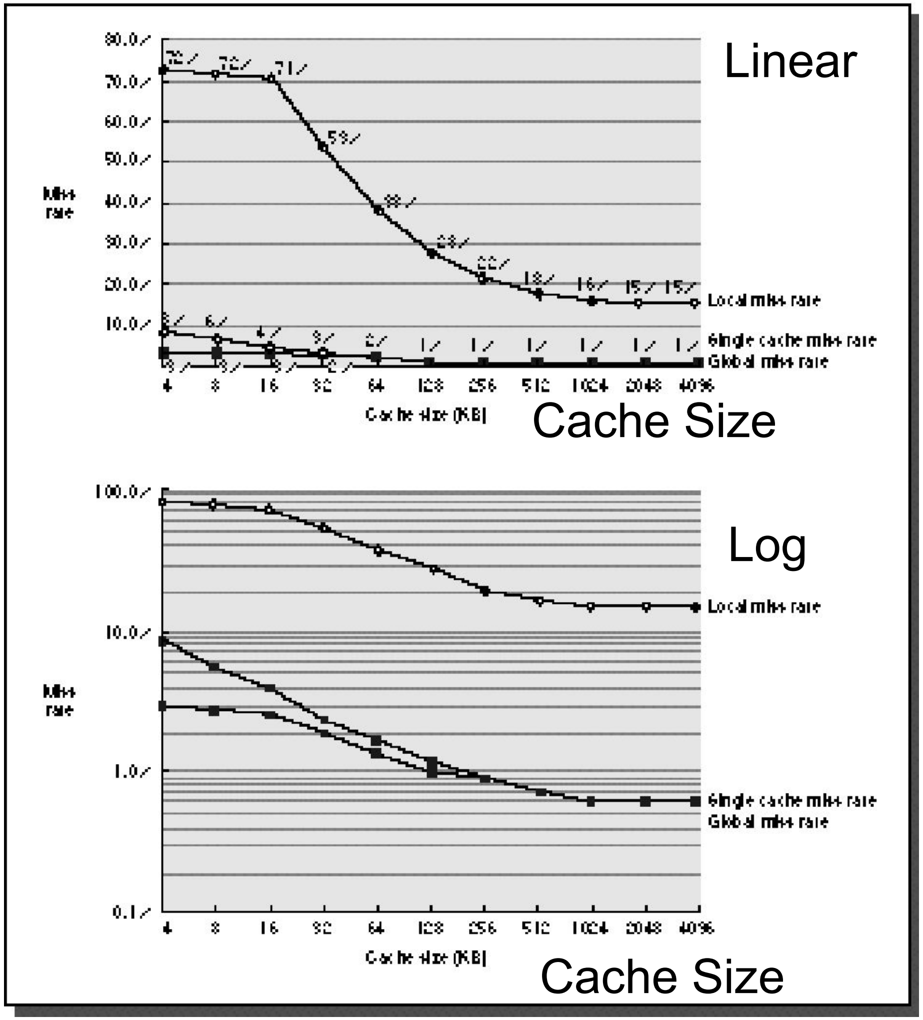
  Miss Penalty$_{L1}$ = Hit Time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$

  AMAT = Hit Time$_{L1}$ +
  Miss Rate$_{L1}$ x (Hit Time$_{L2}$ + Miss Rate$_{L2}$ + Miss Penalty$_{L2}$)

- Definitions:
  - Local miss rate— misses in this cache divided by the total number of memory accesses to this cache (Miss rate$_{L2}$)
  - Global miss rate—misses in this cache divided by the total number of memory accesses generated by the CPU
  - Global Miss Rate is what matters

Comparing Local and Global Miss Rates

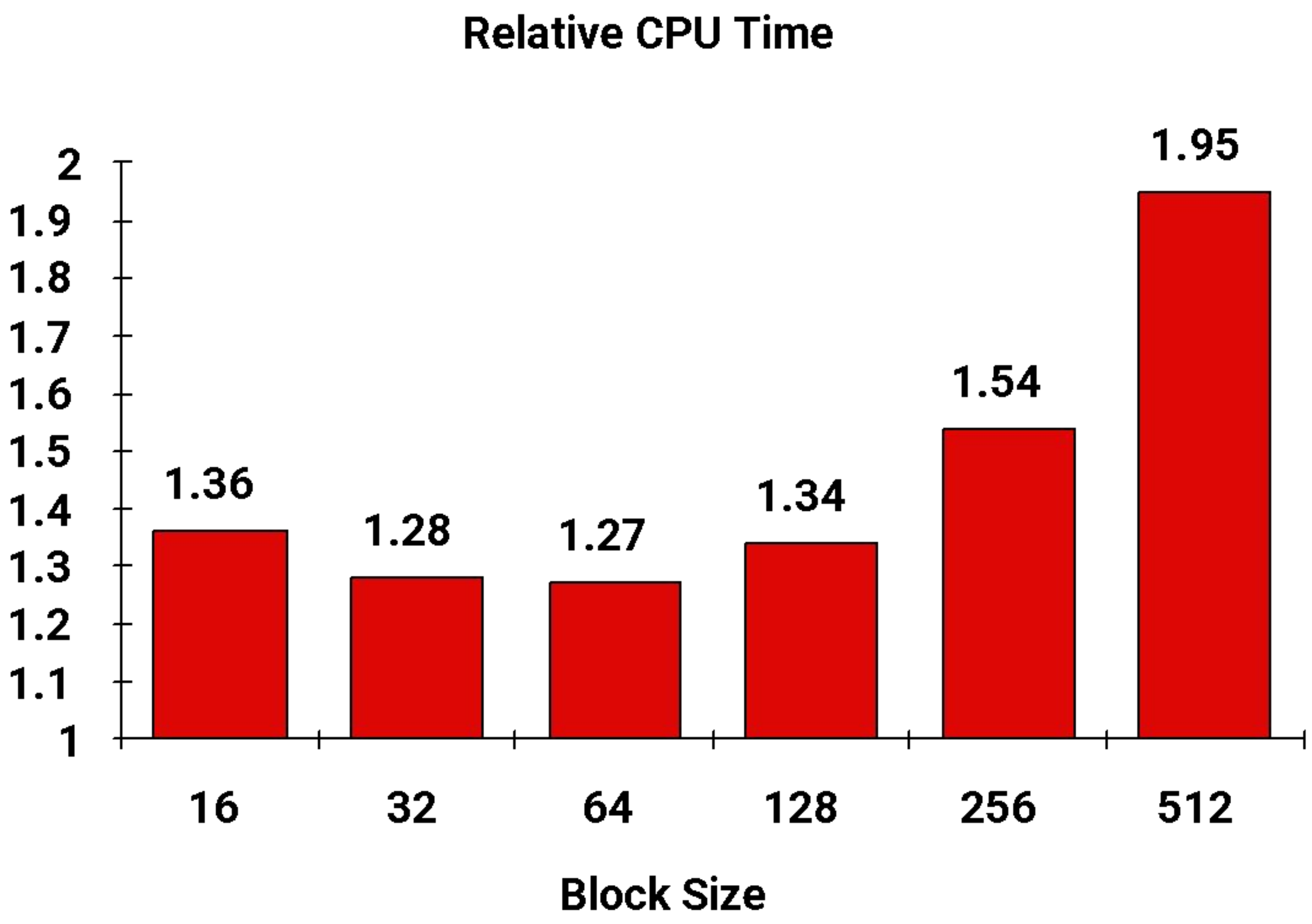
- 32 KByte 1st level cache; Increasing 2nd level cache
- Global miss rate close to single level cache rate provided L2 >> L1
- Don’t use local miss rate
- L2 not tied to CPU clock cycle!
- Cost & A.M.A.T.
- Generally Fast Hit Times and fewer misses
- Since hits are few, target miss reduction



Reducing Misses: Which apply to L2 Cache?

- Reducing Miss Rate
  1. Reduce Misses via Larger Block Size
  2. Reduce Conflict Misses via Higher Associativity
  3. Reducing Conflict Misses via Victim Cache
  4. Reducing Conflict Misses via Pseudo-Associativity
  5. Reducing Misses by HW Prefetching Instr, Data
  6. Reducing Misses by SW Prefetching Data
  7. Reducing Capacity/Conf. Misses by Compiler Optimizations

L2 Cache Block Size & A.M.A.T.



- 32KB L1, 8 byte path to memory

Improving Cache Performance

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

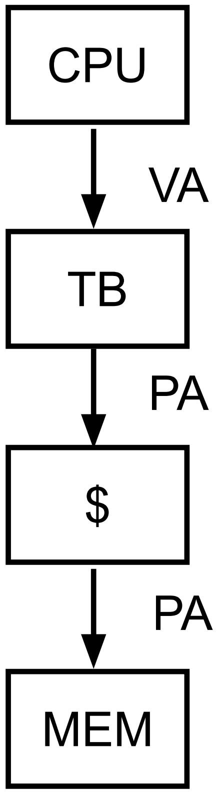
1. Small and Simple Caches

- Why Alpha 21164 has 8KB Instruction and 8KB data cache + 96KB second level cache?
  - Small data cache and clock rate
- Direct Mapped, on chip

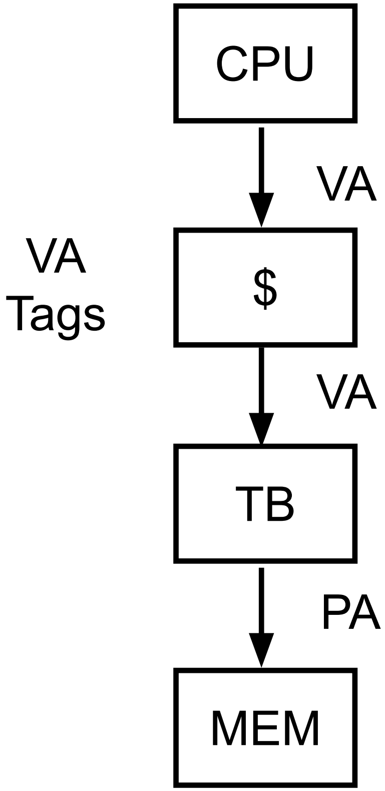
2. Avoiding Address Translation

- Send virtual address to cache? Called Virtually Addressed Cache or just Virtual Cache vs. Physical Cache
  - – Every time process is switched logically must flush the cache; otherwise get false hits
    - Cost is time to flush + "compulsory" misses from empty cache
  - – Dealing with aliases (sometimes called synonyms);
    Two different virtual addresses map to same physical address
  - – I/O must interact with cache, so need virtual address
- Solution to aliases
  - – HW guarantees every cache block has unique physical address
  - – SW guarantee : lower n bits must have same address;
    as long as covers index field & direct mapped, they must be unique; called page coloring
- Solution to cache flush
  - – Add process identifier tag that identifies process as well as address within process: can't get a hit if wrong process

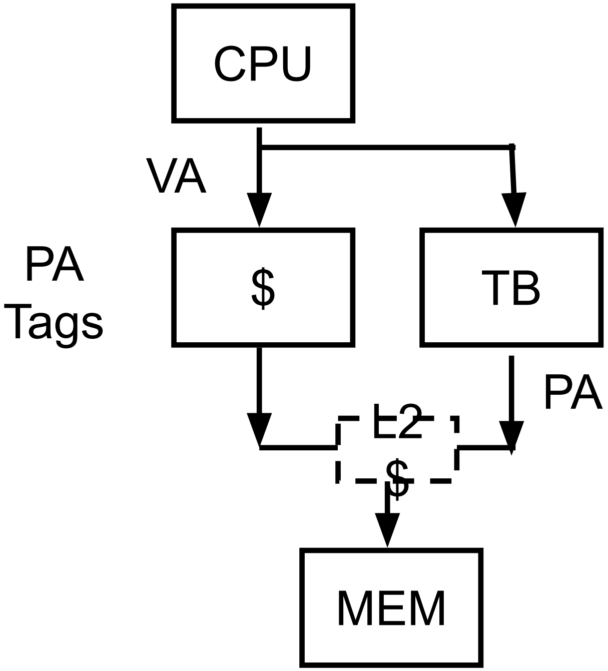
Virtually Addressed Caches



Conventional Organization

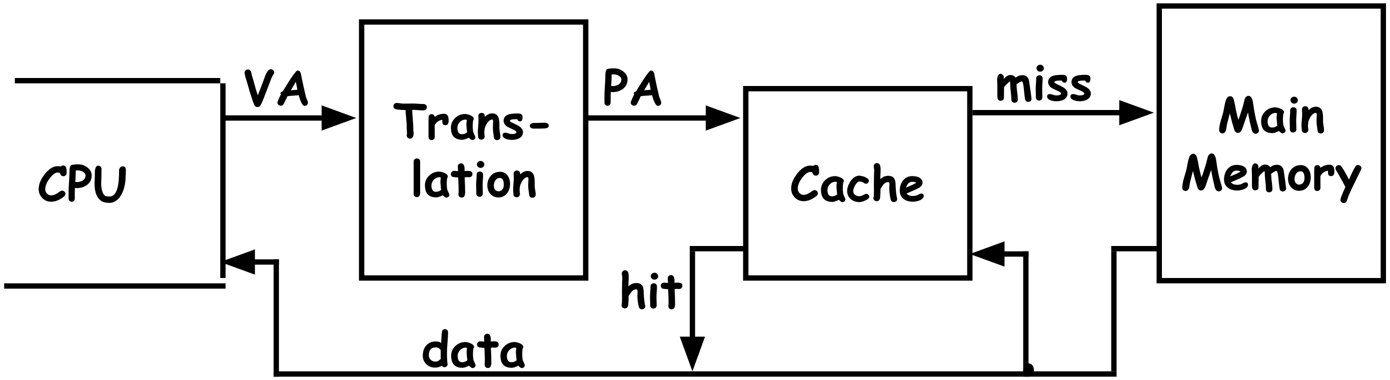


Virtually Addressed Cache
Translate only on miss
Synonym Problem



Overlap $ access with VA translation: requires $ index to remain invariant across translation

Address Translation

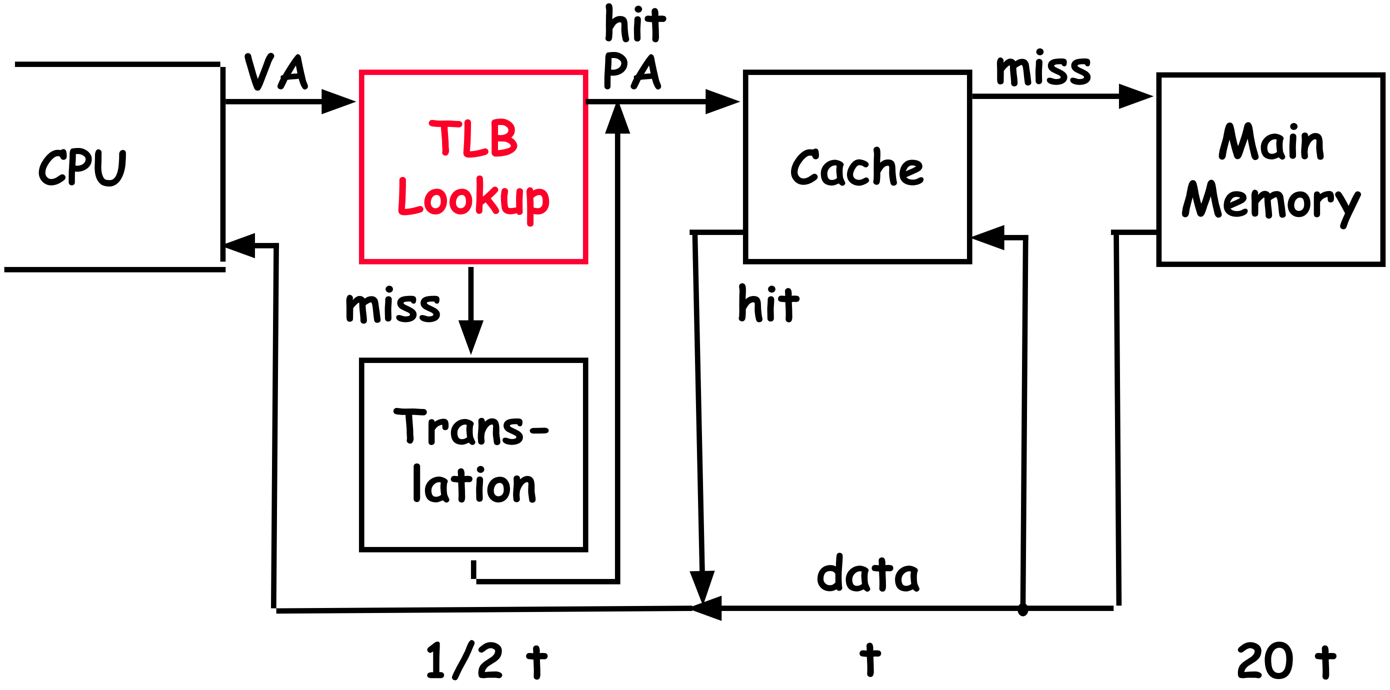


- Page table is a large data structure in memory
- Two memory accesses for every load, store, or instruction fetch!!!
- Virtually addressed cache?
  - synonym problem
- Cache the address translations?
- If index is physical part of address, can start tag access in parallel with translation so that can compare to physical tag

Translation Lookaside Buffers

- Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped
- TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully Associative lookup on these machines.
  - Most mid-range machines use small n-way set associative organizations.

Translation with a TLB



Translation Lookaside Buffer

A way to speed up translation is to use a special cache of recently used page table entries -- this has many names, but the most frequently used is Translation Lookaside Buffer or TLB

Virtual Address	Physical Address	Dirty	Ref	Valid	Access

Really just a cache on the page table mappings

TLB access time comparable to cache access time (much less than main memory access time)

3. Pipelined Writes

- Pipeline Tag Check and Update Cache as separate stages; current write tag check & previous write cache update
- Only STORES in the pipeline; empty during a miss

Store r2, (r1)	Check r1
Add	--
Sub	--
Store r4, (r3)	M[r1]<-r2 & check r3

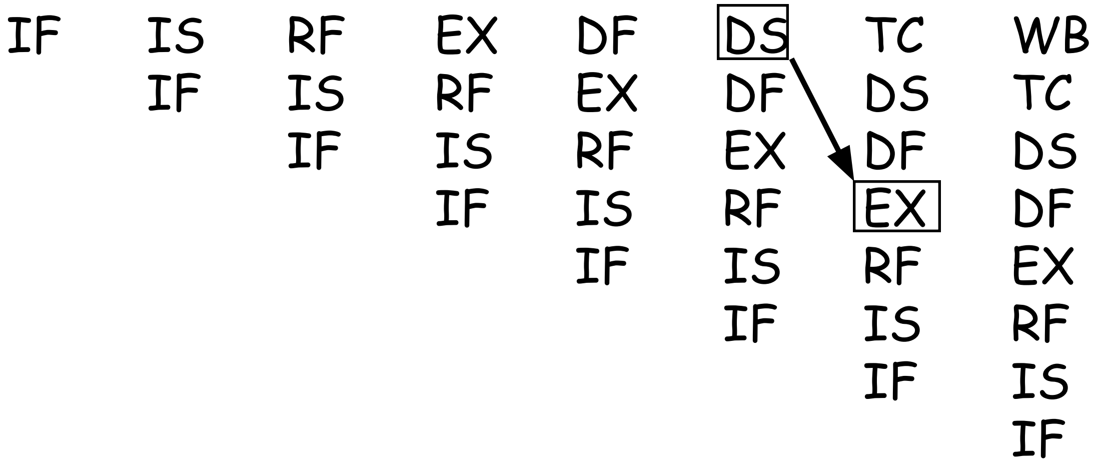
- "Delayed Write Buffer"; must be checked on reads; either complete write or read from buffer

Case Study: MIPS R4000

- 8 Stage Pipeline:
  - IF–first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access.
  - IS–second half of access to instruction cache.
  - RF–instruction decode and register fetch, hazard checking and also instruction cache hit detection.
  - EX–execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation.
  - DF–data fetch, first half of access to data cache.
  - DS–second half of access to data cache.
  - TC–tag check, determine whether the data cache access hit.
  - WB–write back for loads and register-register operations.
- What is impact on Load delay?
  - Need 2 instructions between a load and its use!

Case Study: MIPS R4000

TWO Cycle Load Latency

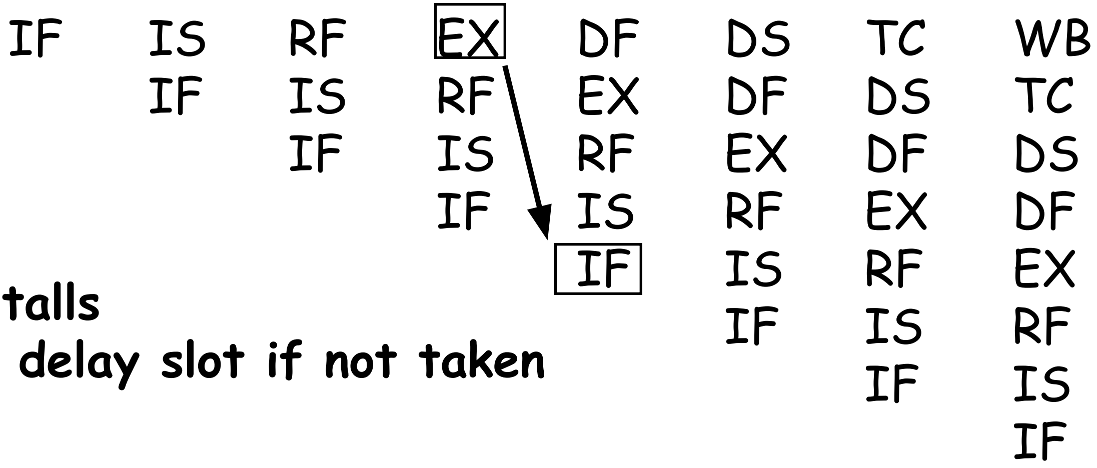


IF	IS	RF	EX	DF	DS	TC	WB
	IF	IS	RF	EX	DF	DS	TC
		IF	IS	RF	EX	DF	DS
			IF	IS	RF	EX	DF
				IF	IS	RF	EX
					IF	IS	RF
						IF	IS
							IF

THREE Cycle Branch Latency

(conditions evaluated during EX phase)

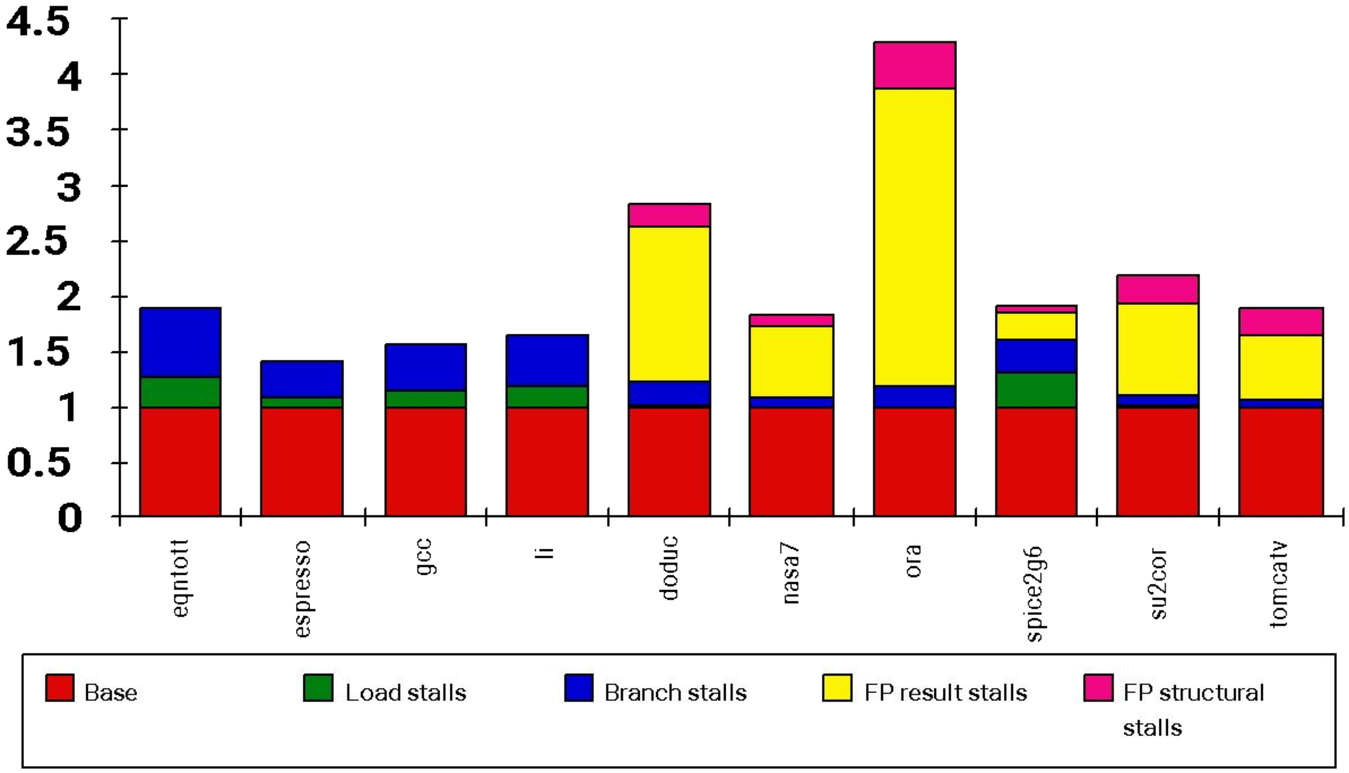
Delay slot plus two stalls
Branch likely cancels delay slot if not taken



IF	IS	RF	EX	DF	DS	TC	WB
	IF	IS	RF	EX	DF	DS	TC
		IF	IS	RF	EX	DF	DS
			IF	IS	RF	EX	DF
				IF	IS	RF	EX
					IF	IS	RF
						IF	IS
							IF

R4000 Performance

- Not ideal CPI of 1:
  - Load stalls (1 or 2 clock cycles)
  - Branch stalls (2 cycles + unfilled slots)
  - FP result stalls: RAW data hazard (latency)
  - FP structural stalls: Not enough FP hardware (parallelism)



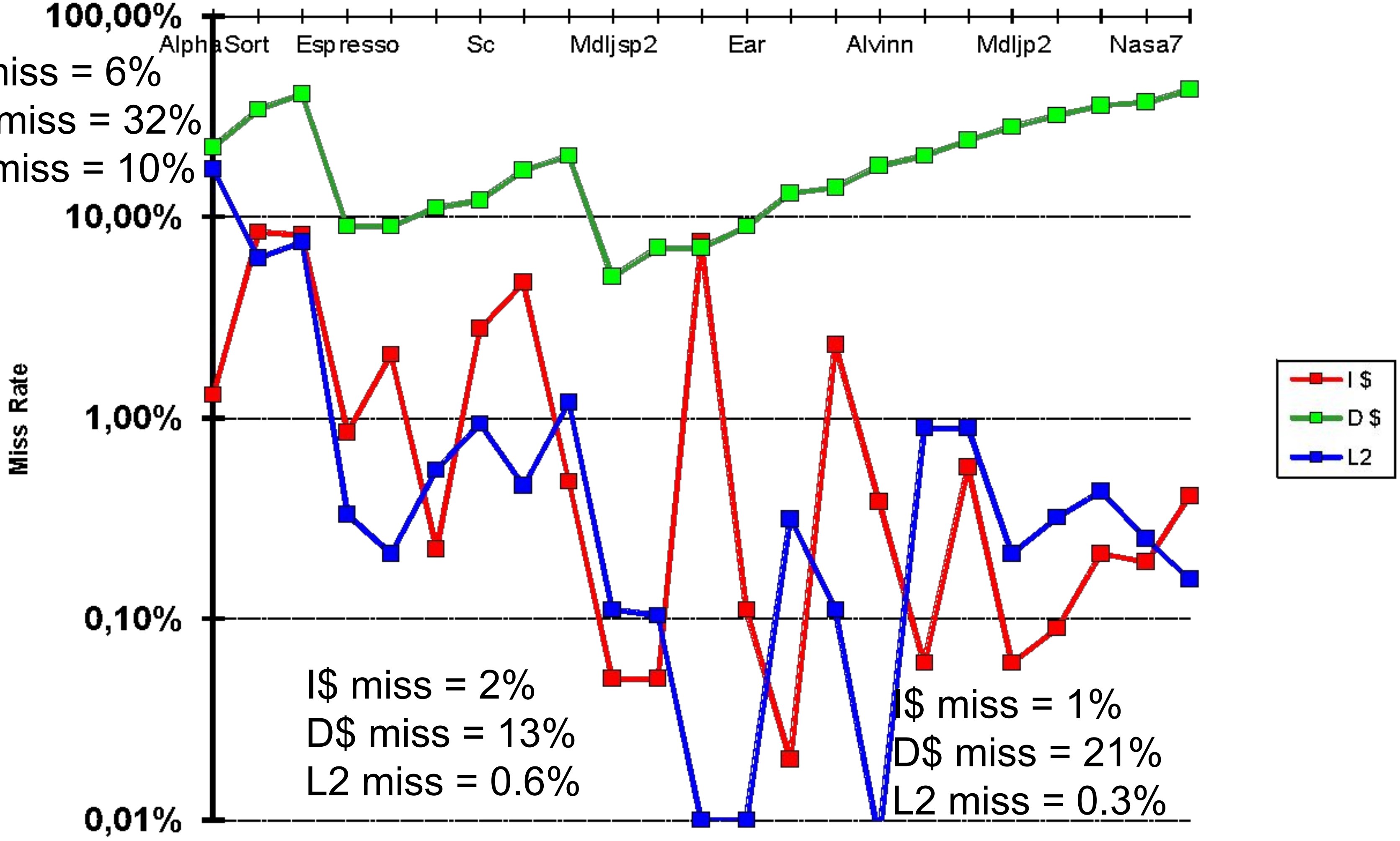
Cache Optimization Summary

	Technique	MR	MP	HT	Complexity
miss rate	Larger Block Size	+	–		0
	Higher Associativity	+		–	1
	Victim Caches	+			2
	Pseudo-Associative Caches	+			2
	HW Prefetching of Instr/Data	+			2
	Compiler Controlled Prefetching	+			3
	Compiler Reduce Misses	+			0
miss penalty	Priority to Read Misses		+		1
	Early Restart & Critical Word 1st		+		2
	Non-Blocking Caches		+		3
	Second Level Caches		+		2
	Better memory system		+		3
hit time	Small & Simple Caches	–		+	0
	Avoiding Address Translation			+	2
	Pipelining Caches			+	2

Cache Cross Cutting Issues

- Superscalar CPU & Number Cache Ports must match: number memory accesses/cycle?
- Speculative Execution and non-faulting option on memory/TLB
- Parallel Execution vs. Cache locality
  - Want far separation to find independent operations vs.. want reuse of data accesses to avoid misses
- I/O and consistency of data between cache and memory
  - Caches => multiple copies of data
  - Consistency by HW or by SW?
  - Where connect I/O to computer?

Alpha Memory Performance: Miss Rates of SPEC92



I$ miss = 6%
D$ miss = 32%
L2 miss = 10%

I$ miss = 2%
D$ miss = 13%
L2 miss = 0.6%

I$ miss = 1%
D$ miss = 21%
L2 miss = 0.3%

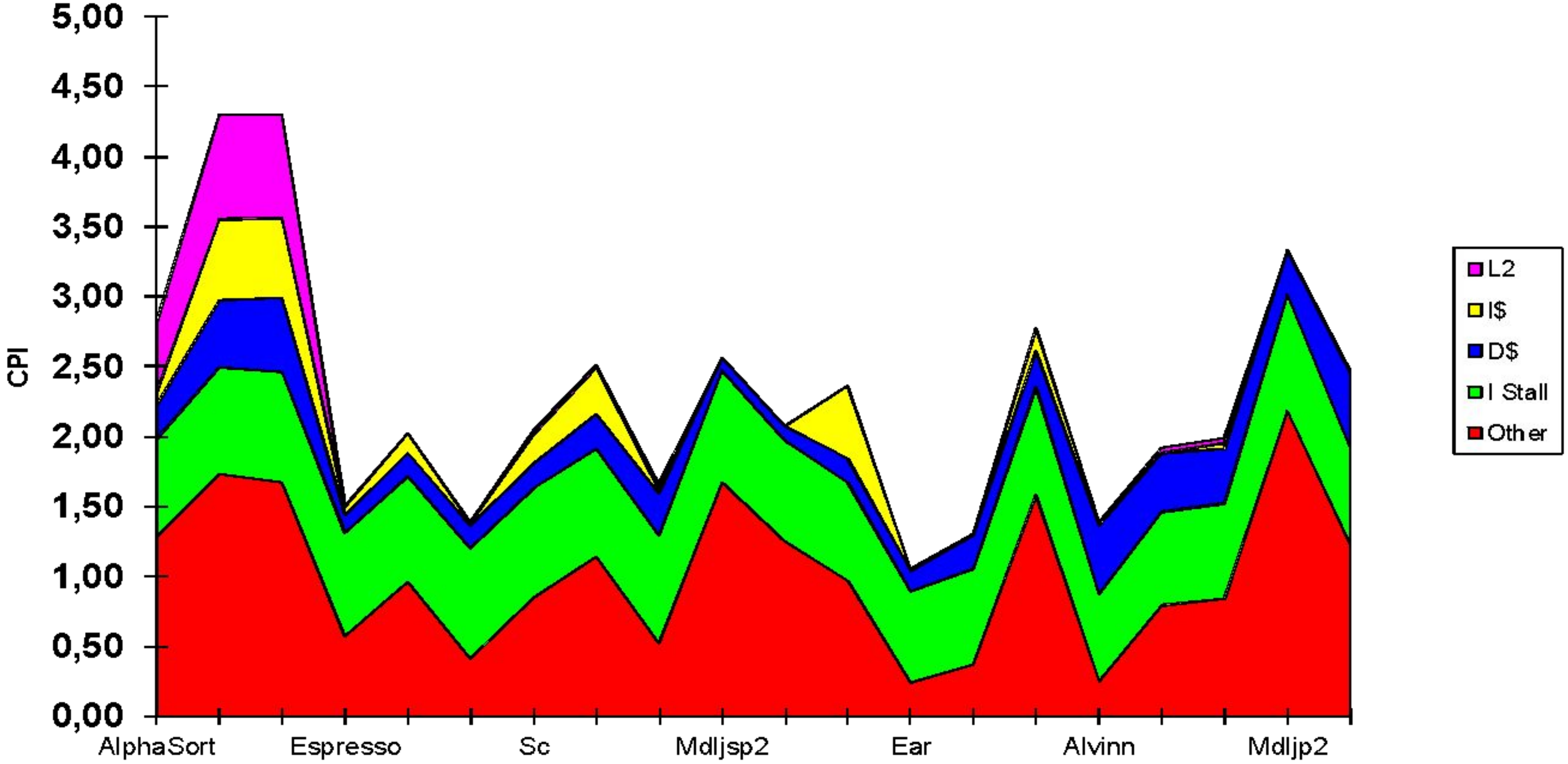
8K

8K

2M

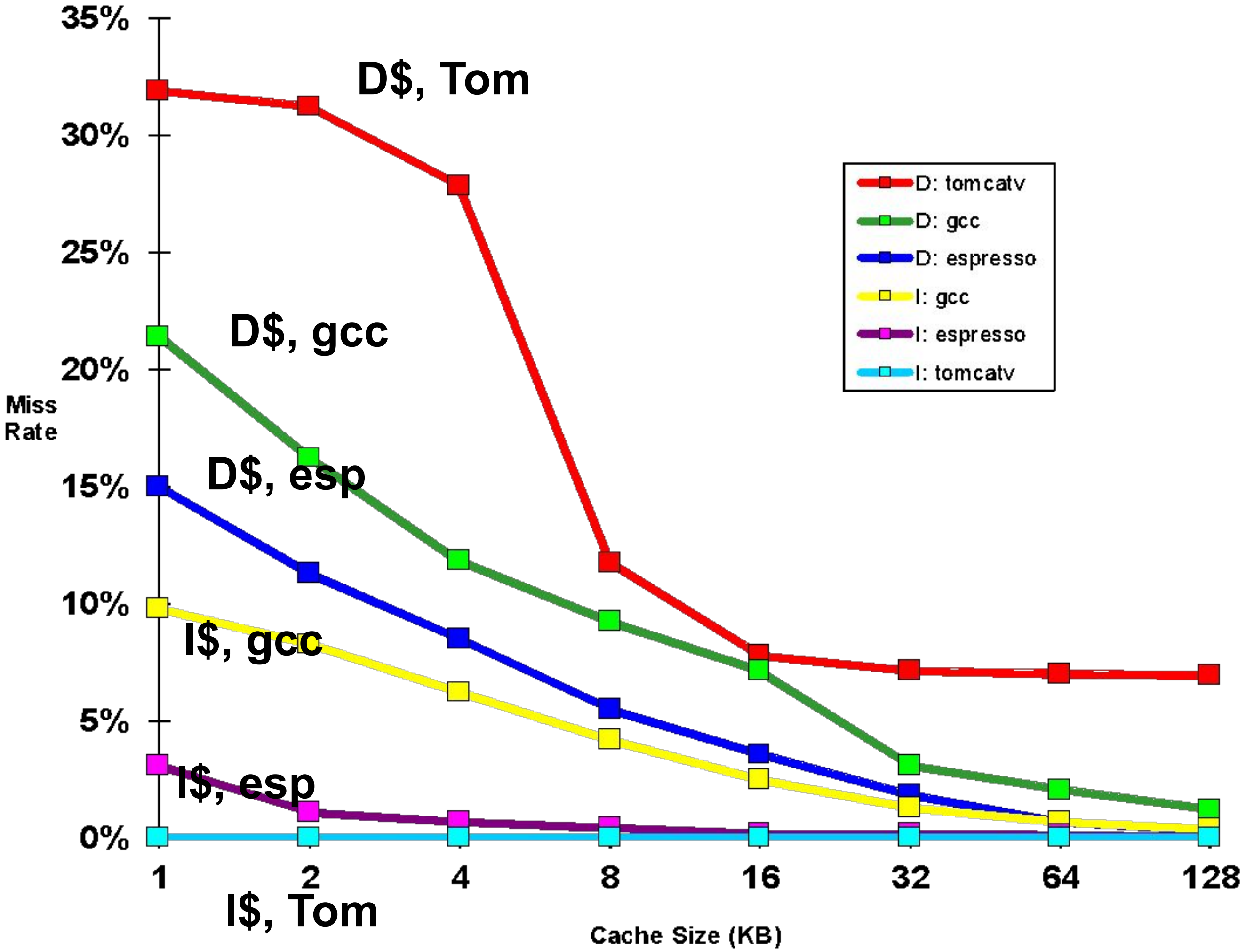
Alpha CPI Components

- Instruction stall: branch mispredict (green);
- Data cache (blue); Instruction cache (yellow); L2$ (pink) Other: compute + reg conflicts, structural conflicts



Predicting Cache Performance from Different Prog. (ISA, compiler, ...)

- 4KB Data cache miss rate 8%,12%, or 28%?
- 1KB Instr cache miss rate 0%,3%,or 10%?
- Alpha vs.. MIPS for 8KB Data $: 17% vs. 10%
- Why 2X Alpha v. MIPS?



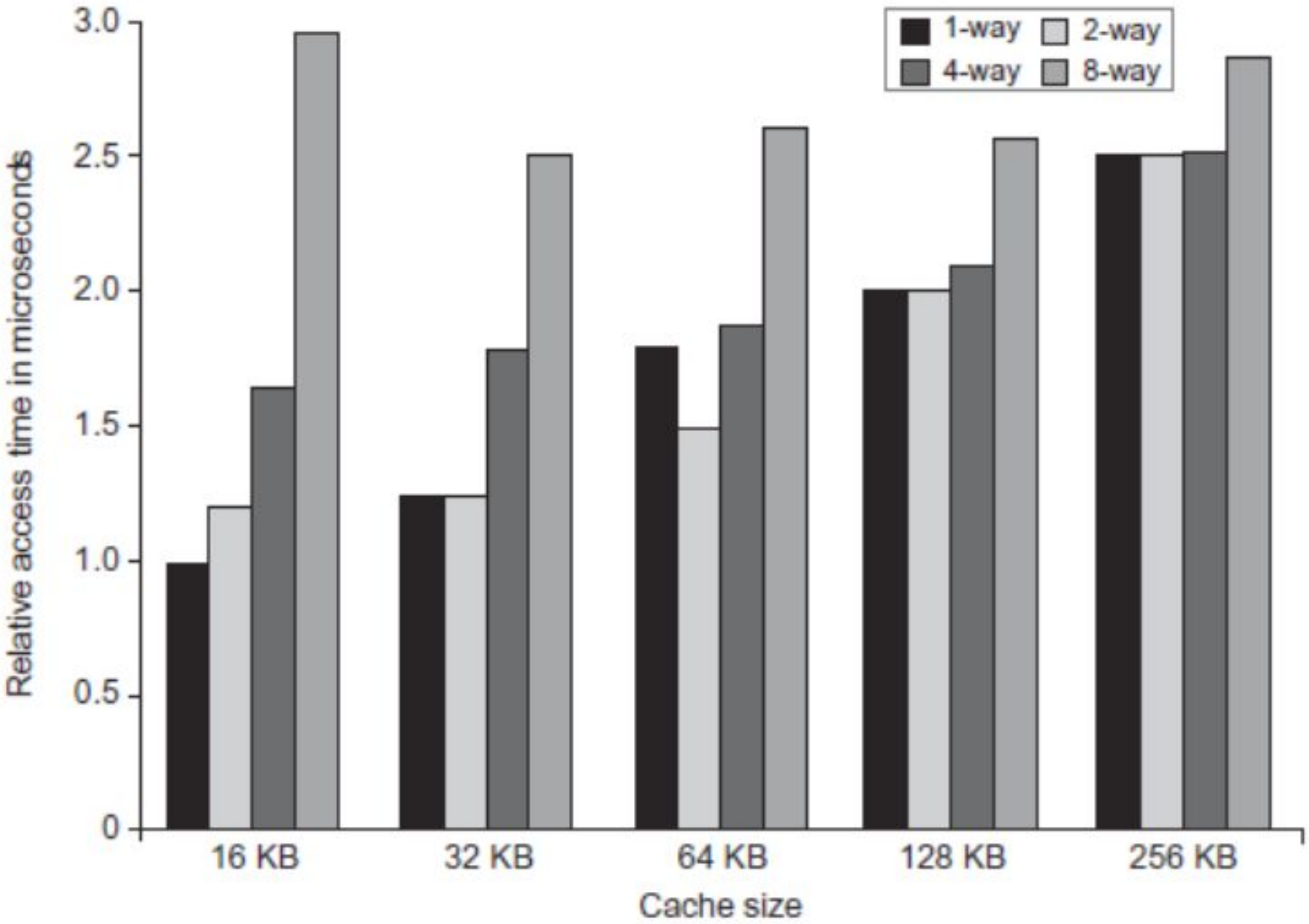
Advanced Optimizations

- Reduce hit time
  - Small and simple first-level caches
  - Way prediction
- Increase bandwidth
  - Pipelined caches, multibanked caches, non-blocking caches
- Reduce miss penalty
  - Critical word first, merging write buffers
- Reduce miss rate
  - Compiler optimizations
- Reduce miss penalty or miss rate via parallelization
  - Hardware or compiler prefetching

Advanced Optimizations

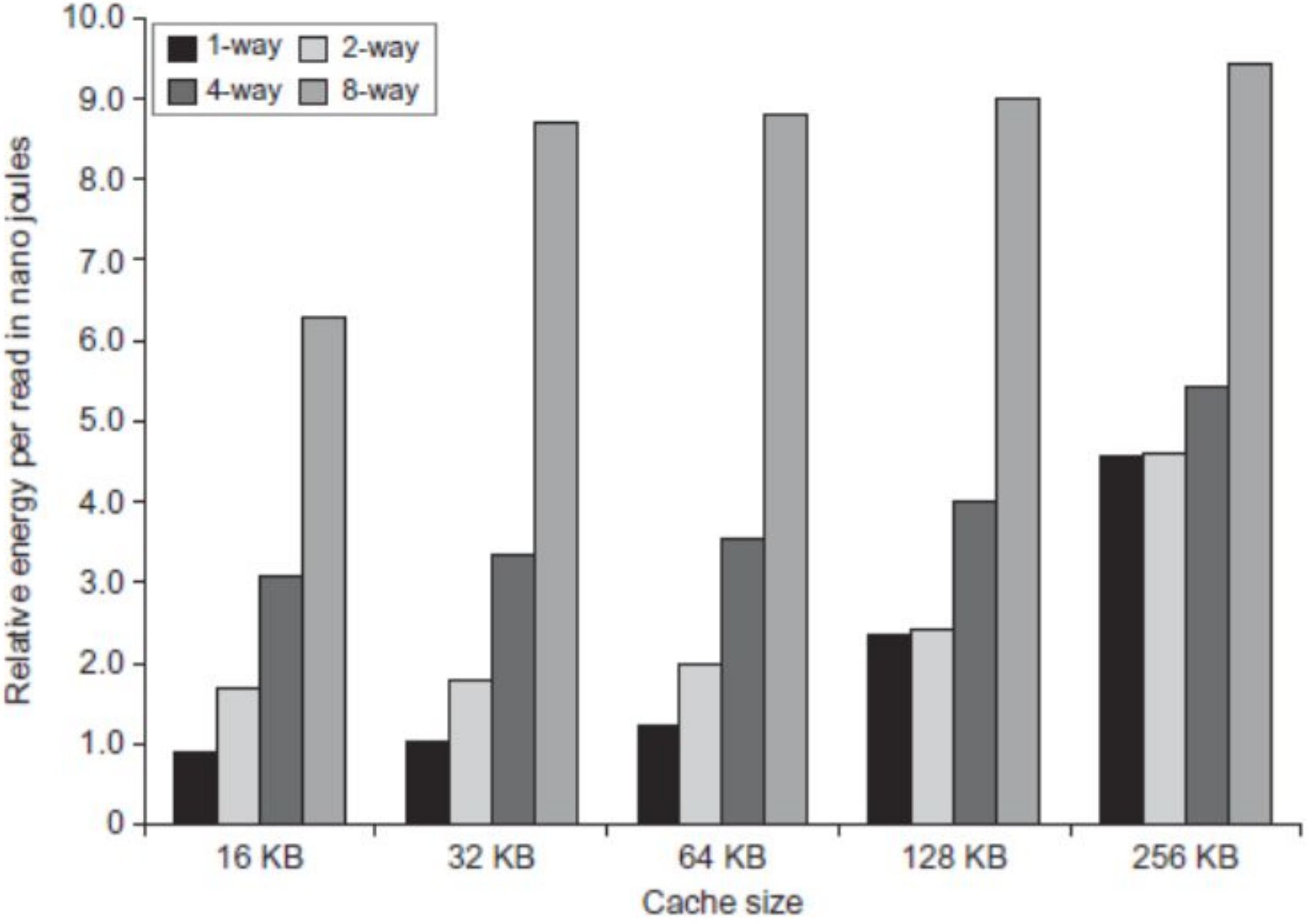
- Small and simple first level caches
  - Critical timing path:
    - addressing tag memory, then
    - comparing tags, then
    - selecting correct set
  - Direct-mapped caches can overlap tag compare and transmission of data
  - Lower associativity reduces power because fewer cache lines are accessed

L1 Size and Associativity



Access time vs. size and associativity

L1 Size and Associativity



Energy per read vs. size and associativity

Way Prediction

- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - “Way selection”
  - Increases mis-prediction penalty

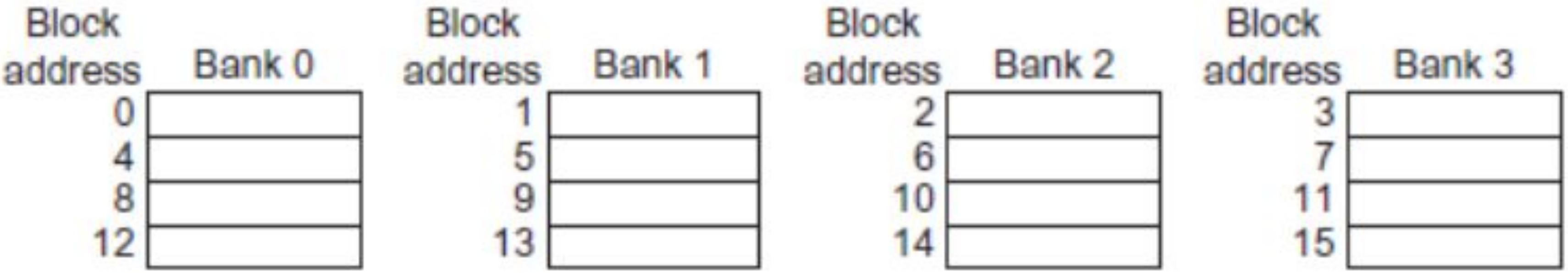
Pipelined Caches

- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles
- Increases branch mis-prediction penalty
- Makes it easier to increase associativity

Multibanked Caches

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2

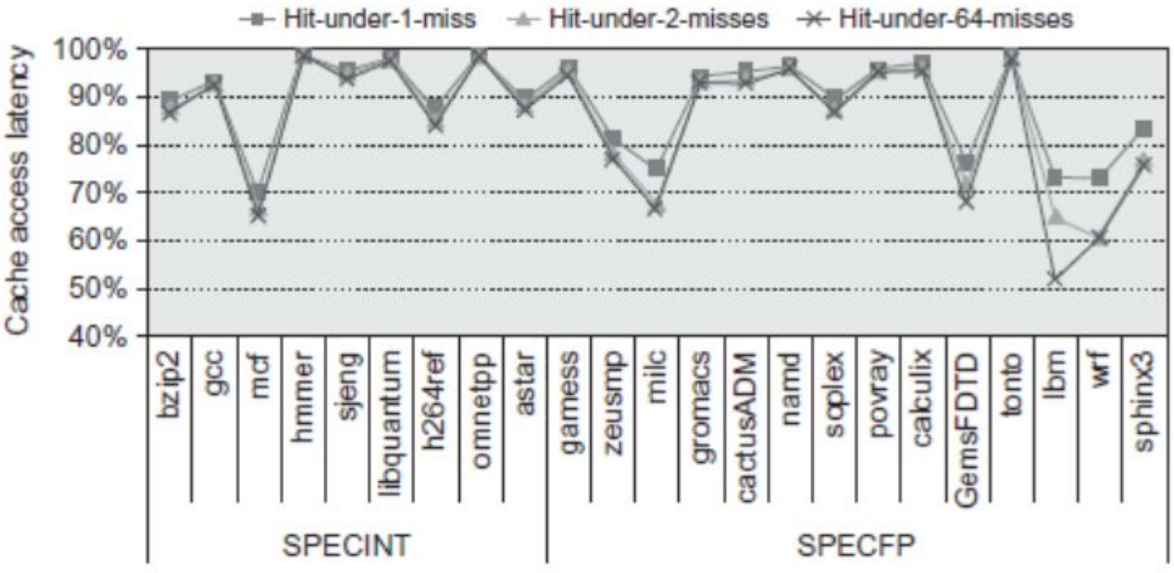
- Interleave banks according to block address



Block address	Bank 0	Block address	Bank 1	Block address	Bank 2	Block address	Bank 3
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	

Nonblocking Caches

- Allow hits before previous misses complete
  - "Hit under miss"
  - "Hit under multiple miss"
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty



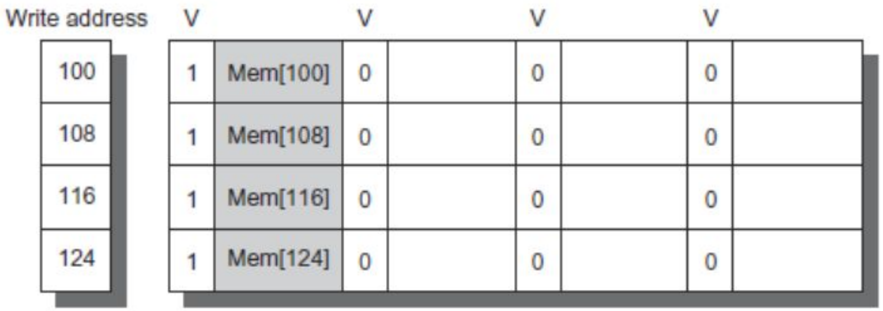
Critical Word First, Early Restart

- Critical word first
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
- Early restart
  - Request words in normal order
  - Send missed work to the processor as soon as it arrives

- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

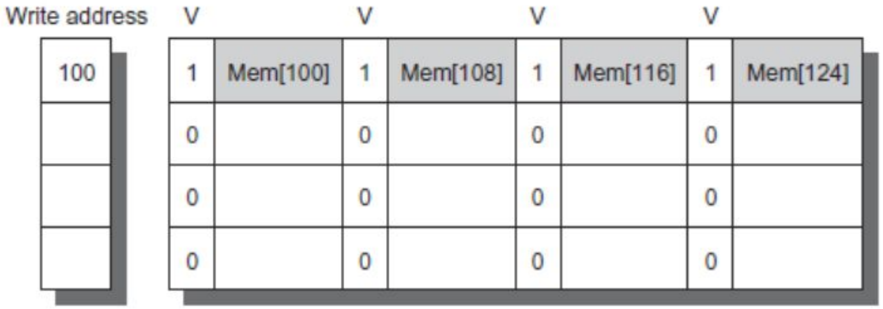
Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Do not apply to I/O addresses



Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

No write buffering



Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

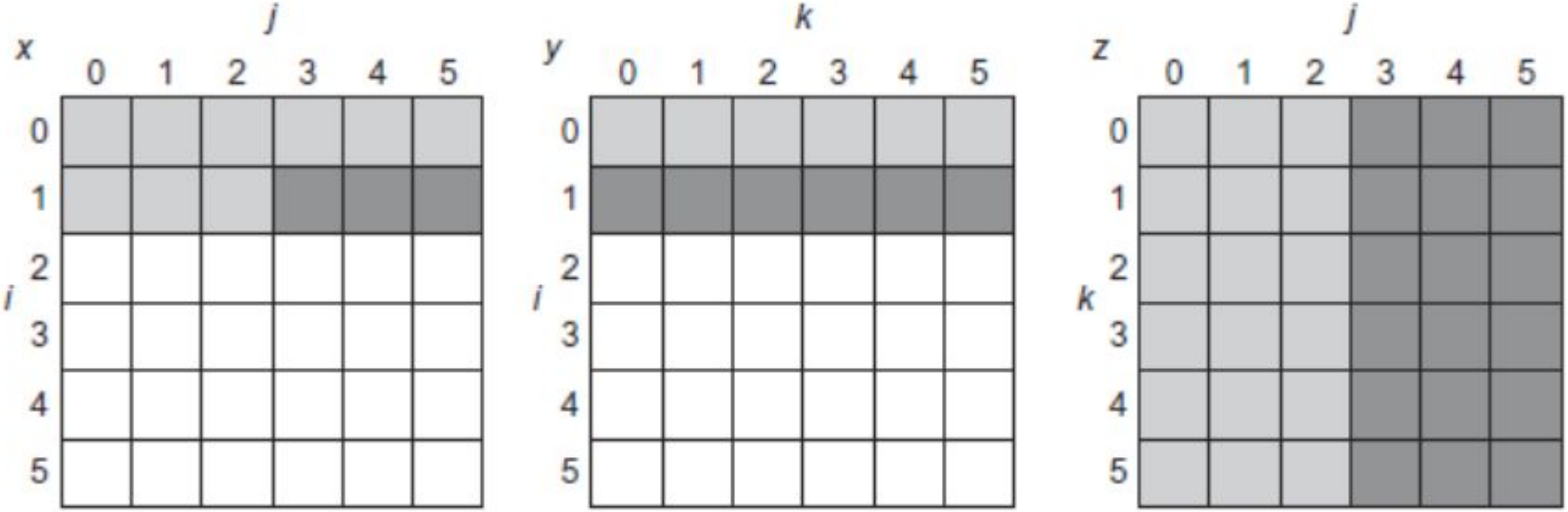
Write buffering

Compiler Optimizations

- Loop Interchange
  - Swap nested loops to access memory in sequential order
- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

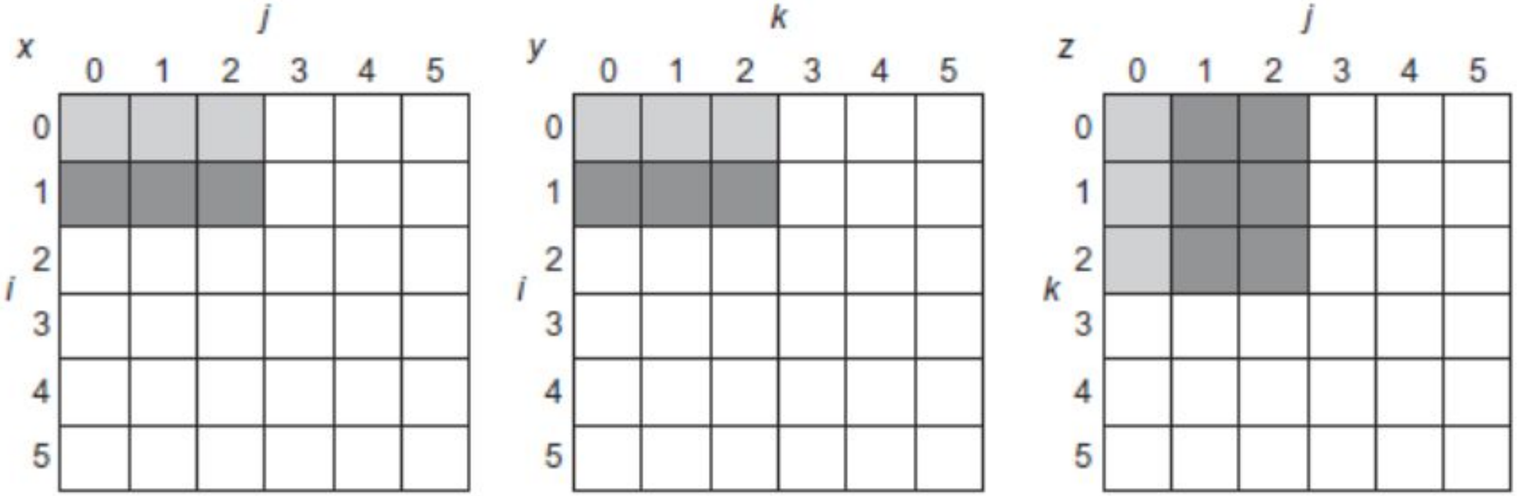
Blocking

```
for (i = 0; i < N; i = i + 1)
  for (j = 0; j < N; j = j + 1)
  {
    r = 0;
    for (k = 0; k < N; k = k + 1)
      r = r + y[i][k]*z[k][j];
    x[i][j] = r;
};
```



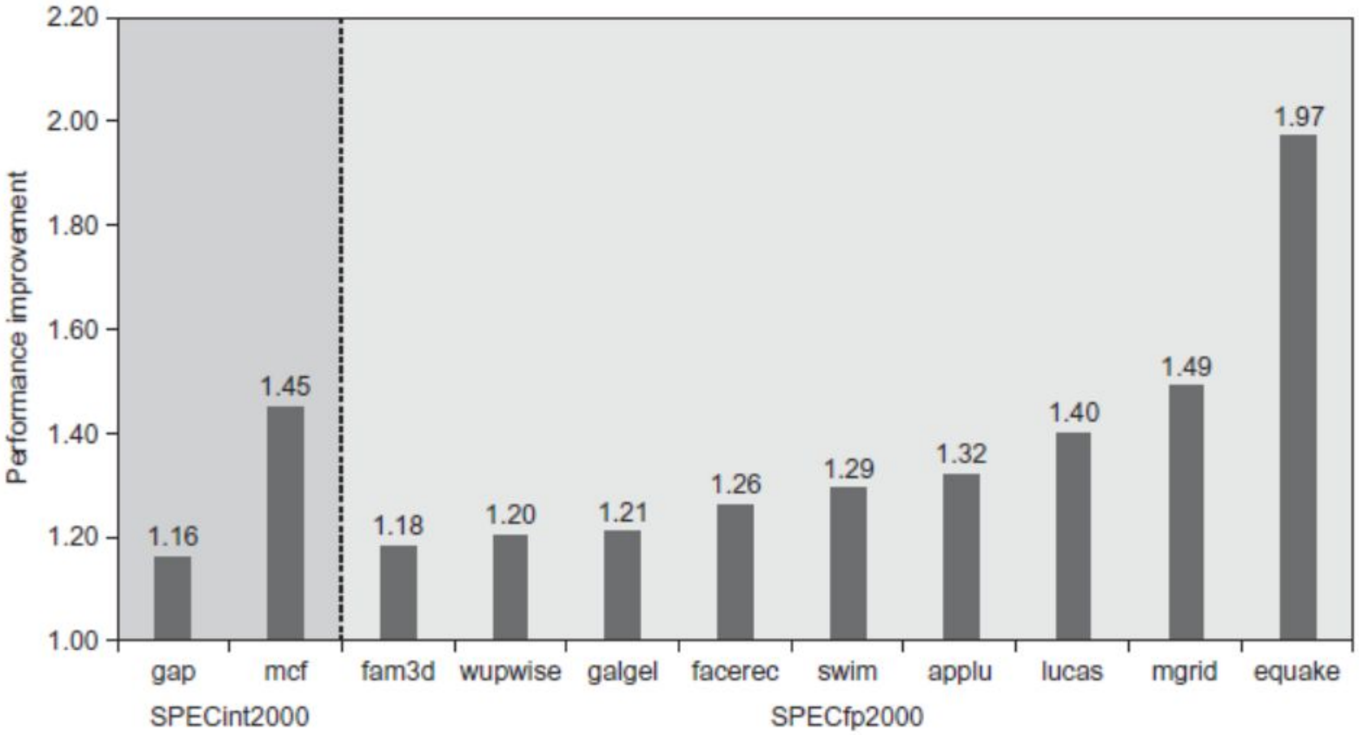
Blocking

```
for (jj = 0; jj < N; jj = jj + B)
  for (kk = 0; kk < N; kk = kk + B)
    for (i = 0; i < N; i = i + 1)
      for (j = jj; j < min(jj + B,N); j = j + 1)
      {
        r = 0;
        for (k = kk; k < min(kk + B,N); k = k + 1)
          r = r + y[i][k]*z[k][j];
        x[i][j] = x[i][j] + r;
};
```



Hardware Prefetching

- Fetch two blocks on miss (include next sequential block)



Pentium 4 Pre-fetching

Compiler Prefetching

- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn’t cause exceptions

- Register prefetch
  - Loads data into register
- Cache prefetch
  - Loads data into cache

- Combine with loop unrolling and software pipelining

Use HBM to Extend Hierarchy

- 128 MiB to 1 GiB
- Smaller blocks require substantial tag storage
- Larger blocks are potentially inefficient

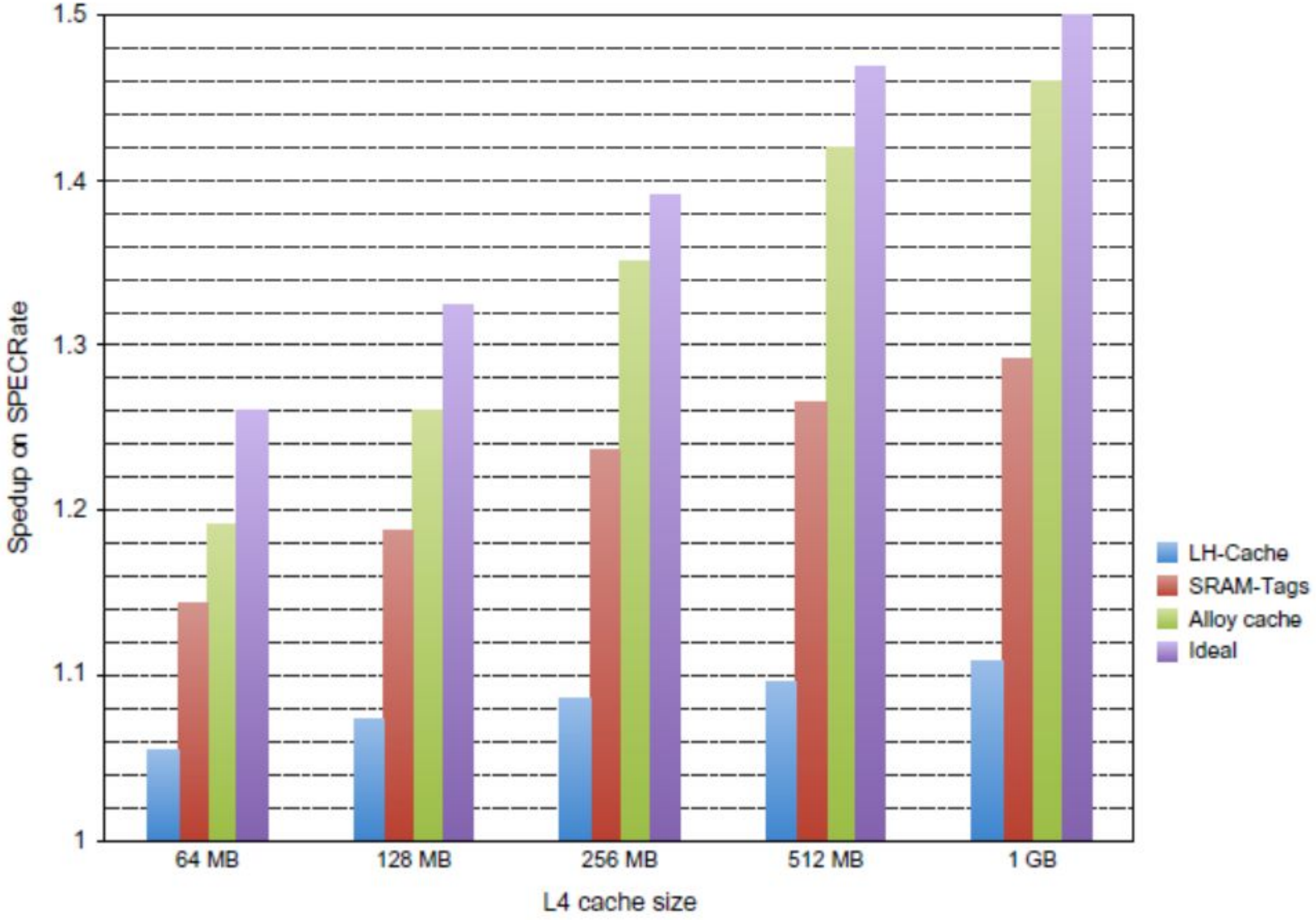
- One approach (L-H):
  - Each SDRAM row is a block index
  - Each row contains set of tags and 29 data segments
  - 29-set associative
  - Hit requires a CAS

Use HBM to Extend Hierarchy

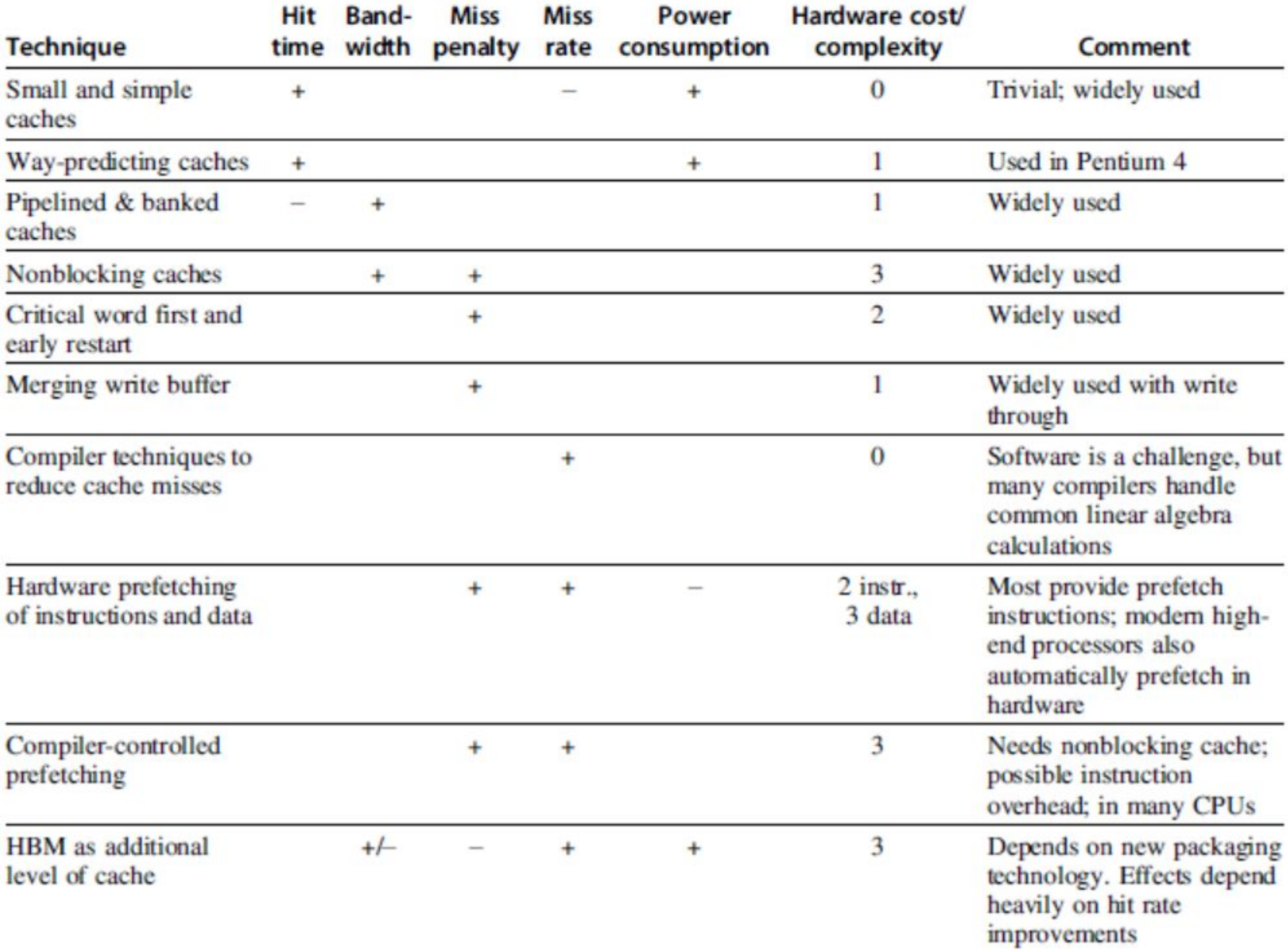
- Another approach (Alloy cache):
  - Mold tag and data together
  - Use direct mapped

- Both schemes require two DRAM accesses for misses
  - Two solutions:
    - Use map to keep track of blocks
    - Predict likely misses

Use HBM to Extend Hierarchy

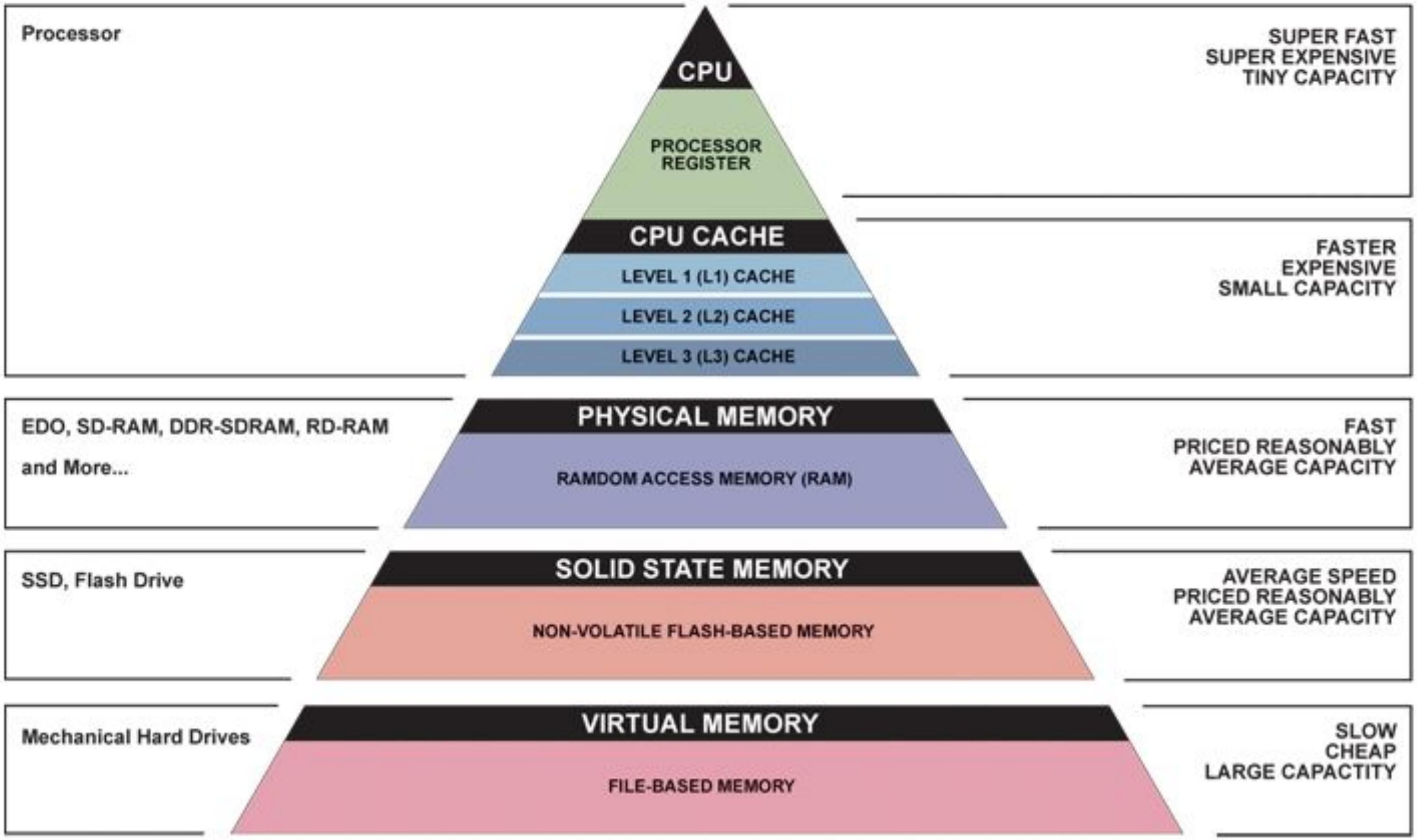


Summary



Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined & banked caches	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs
HBM as additional level of cache		+/–	–	+	+	3	Depends on new packaging technology. Effects depend heavily on hit rate improvements

Computer Memory Hierarchy



Processor

CPU

PROCESSOR REGISTER

SUPER FAST
SUPER EXPENSIVE
TINY CAPACITY

CPU CACHE

LEVEL 1 (L1) CACHE

LEVEL 2 (L2) CACHE

LEVEL 3 (L3) CACHE

FASTER
EXPENSIVE
SMALL CAPACITY

EDO, SD-RAM, DDR-SDRAM, RD-RAM

and More...

PHYSICAL MEMORY

RAMDOM ACCESS MEMORY (RAM)

FAST
PRICED REASONABLY
AVERAGE CAPACITY

SSD, Flash Drive

SOLID STATE MEMORY

NON-VOLATILE FLASH-BASED MEMORY

AVERAGE SPEED
PRICED REASONABLY
AVERAGE CAPACITY

Mechanical Hard Drives

VIRTUAL MEMORY

FILE-BASED MEMORY

SLOW
CHEAP
LARGE CAPACTITY

▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

http://www.bit-tech.net/hardware/memory/2007/11/15/the_secrets_of_pc_memory_part_1/3

Memory Technology and Optimizations

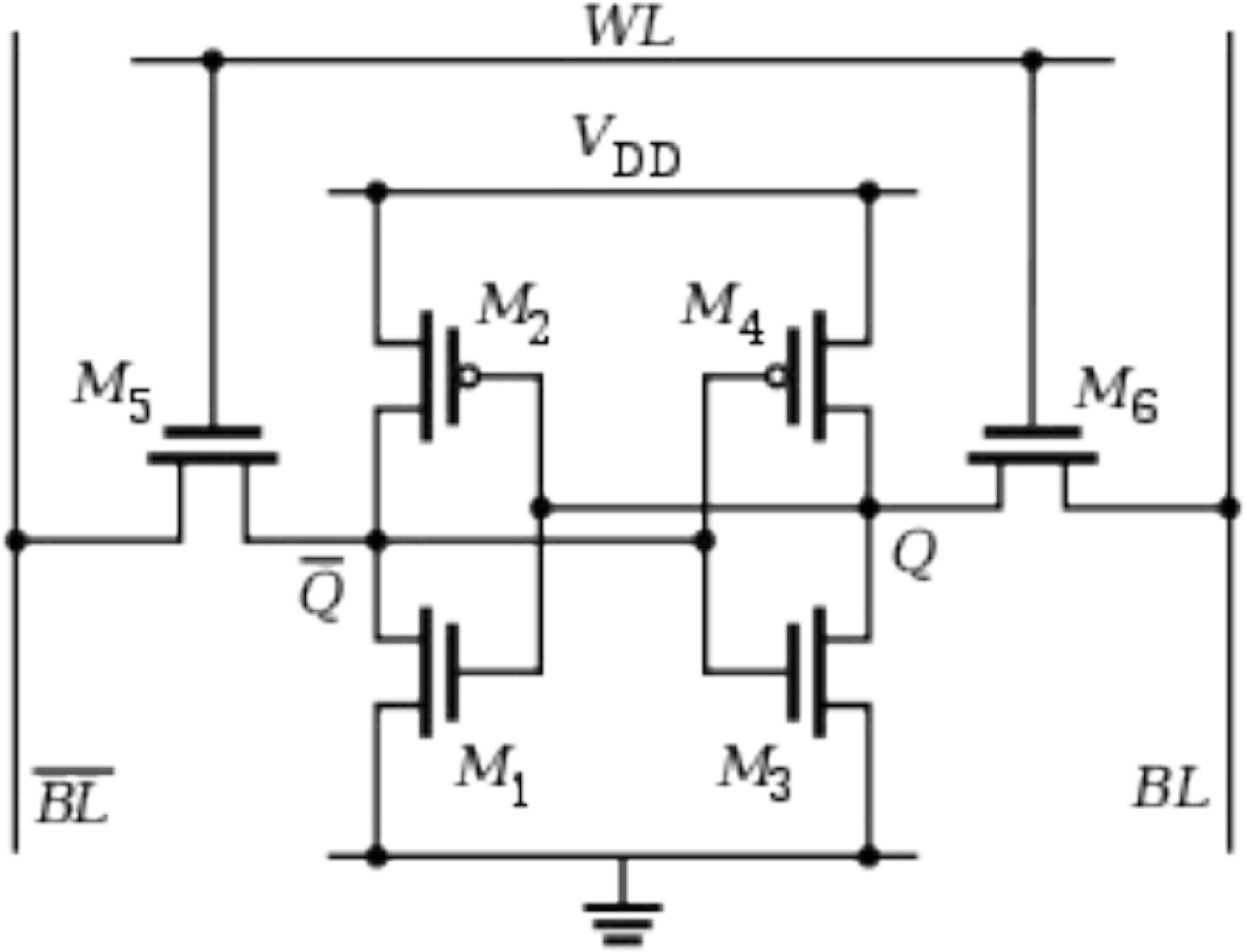
- Performance metrics
  - Latency is concern of cache
  - Bandwidth is concern of multiprocessors and I/O
  - Access time
    - Time between read request and when desired word arrives
  - Cycle time
    - Minimum time between unrelated requests to memory

- SRAM memory has low latency, use for cache
- Organize DRAM chips into many banks for high bandwidth, use for main memory

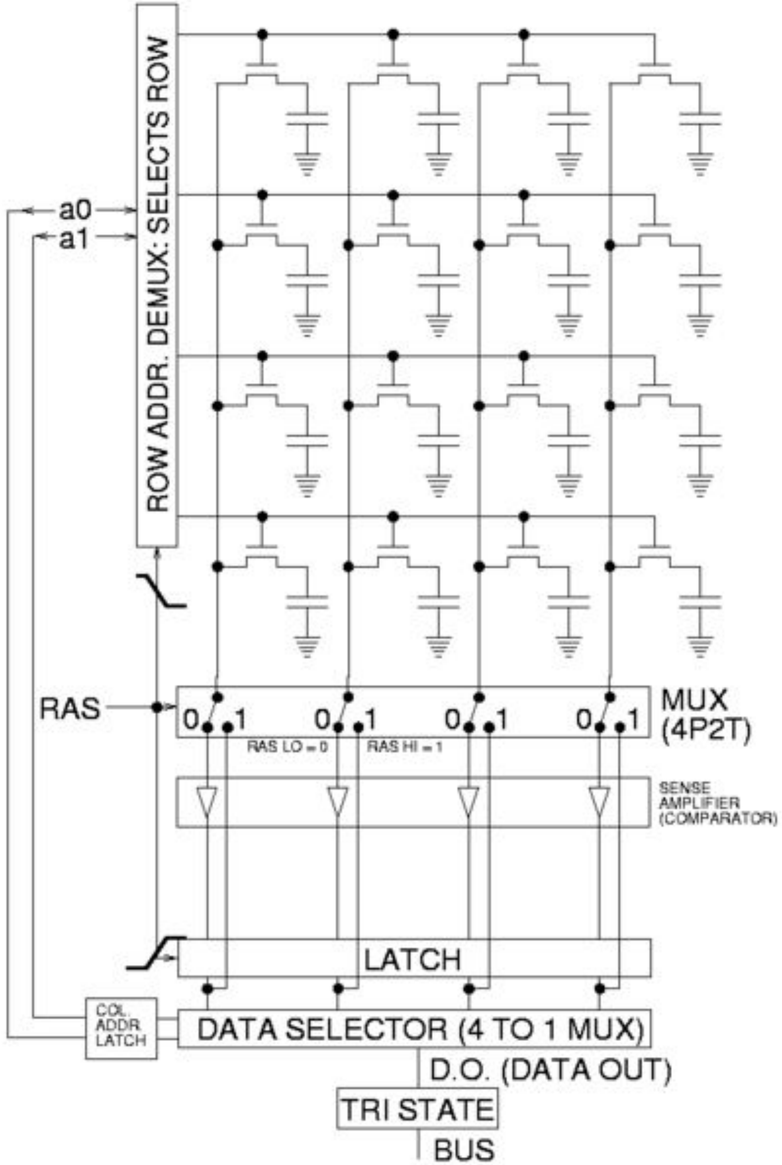
Memory Technology

- SRAM
  - Requires low power to retain bit
  - Requires 6 transistors/bit
- DRAM
  - Must be re-written after being read
  - Must also be periodically refeshed
    - Every ~ 8 ms (roughly 5% of time)
    - Each row can be refreshed simultaneously
  - One transistor/bit
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

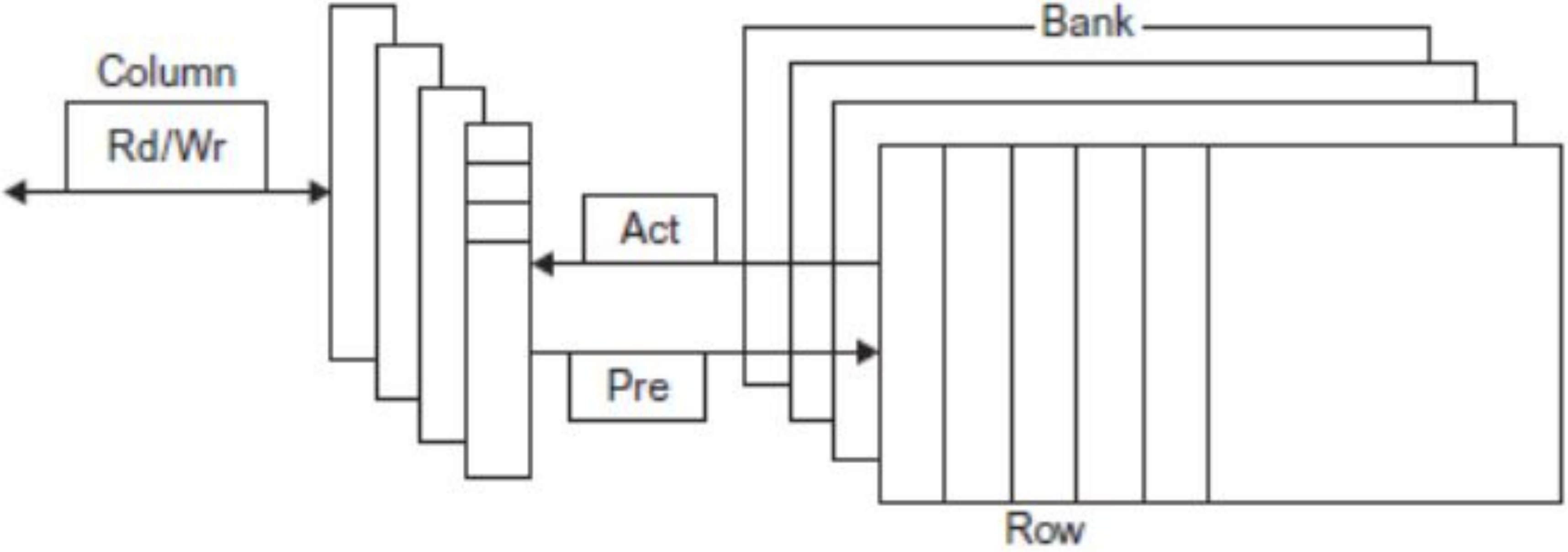
A SRAM Example



A DRAM Example



Internal Organization of DRAM

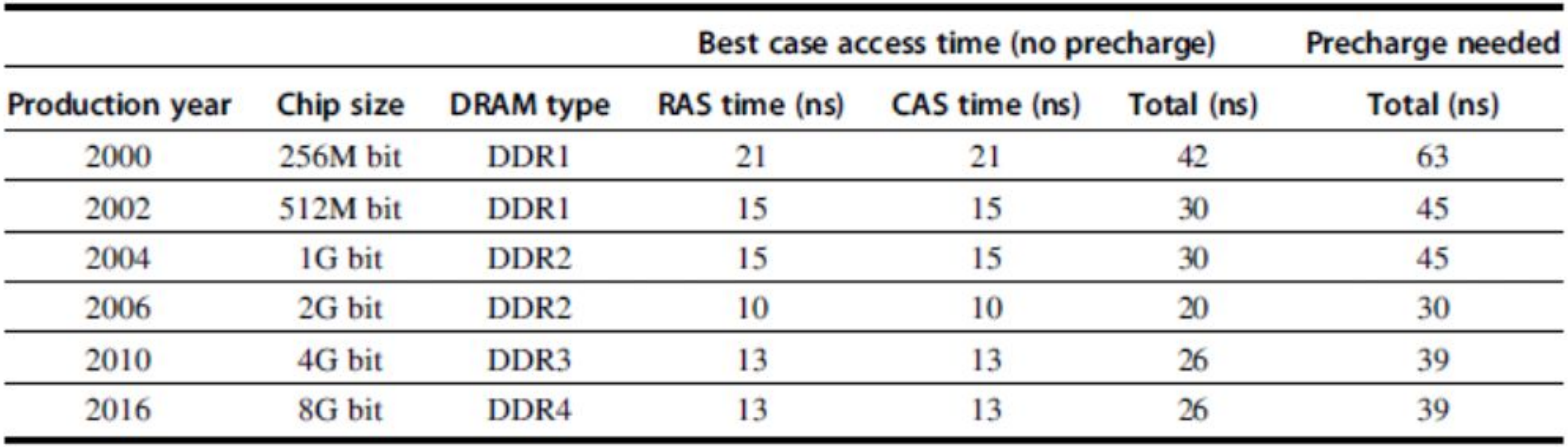


Memory Technology

- Amdahl:
  - Memory capacity should grow linearly with processor speed
  - Unfortunately, memory capacity and speed has not kept pace with processors

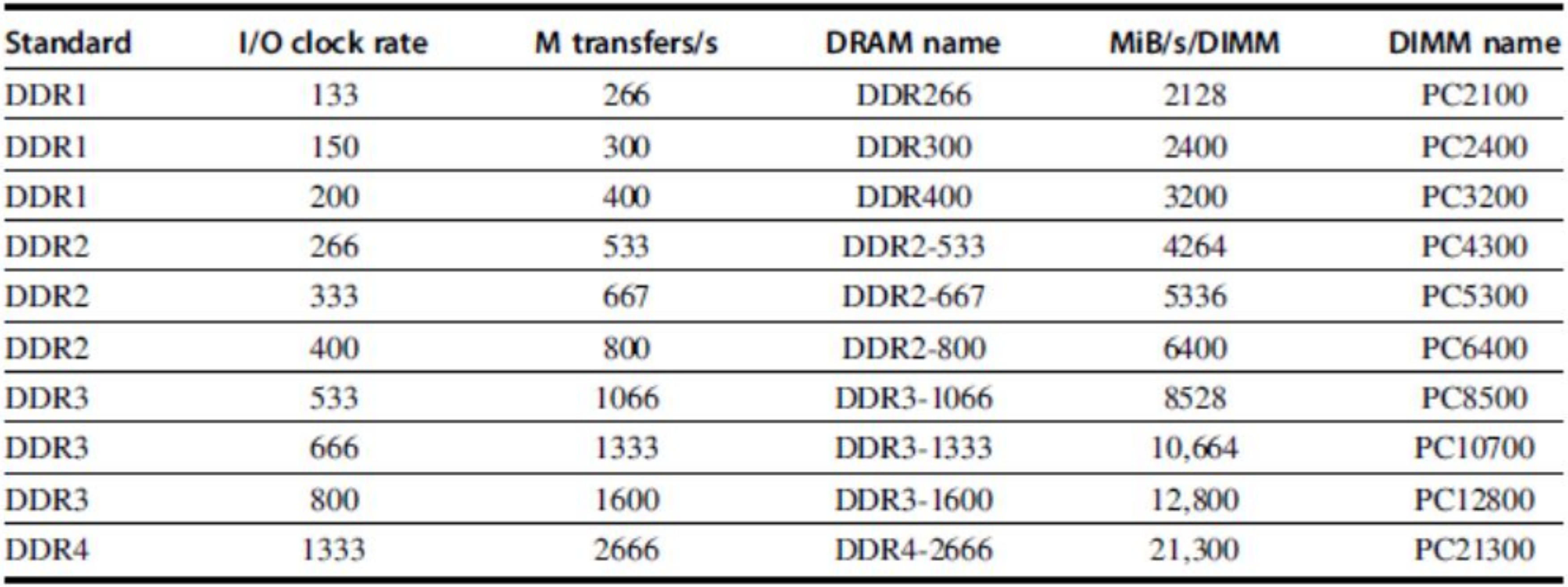
- Some optimizations:
  - Multiple accesses to same row
  - Synchronous DRAM
    - Added clock to DRAM interface
    - Burst mode with critical word first
  - Wider interfaces
  - Double data rate (DDR)
  - Multiple banks on each DRAM device

Memory Optimizations



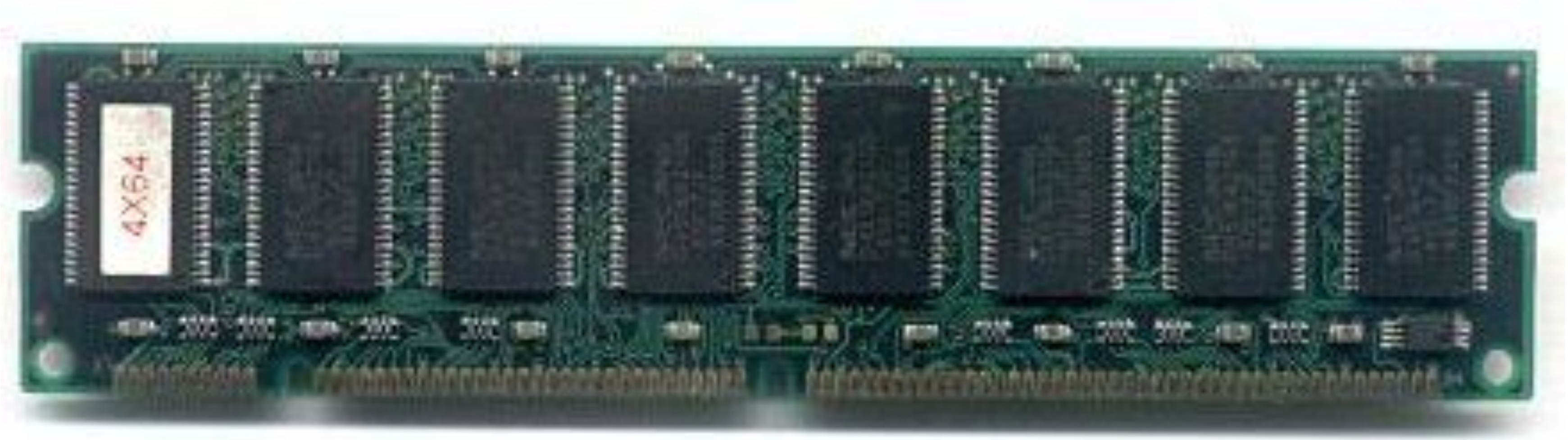
			Best case access time (no precharge)			Precharge needed
Production year	Chip size	DRAM type	RAS time (ns)	CAS time (ns)	Total (ns)	Total (ns)
2000	256M bit	DDR1	21	21	42	63
2002	512M bit	DDR1	15	15	30	45
2004	1G bit	DDR2	15	15	30	45
2006	2G bit	DDR2	10	10	20	30
2010	4G bit	DDR3	13	13	26	39
2016	8G bit	DDR4	13	13	26	39

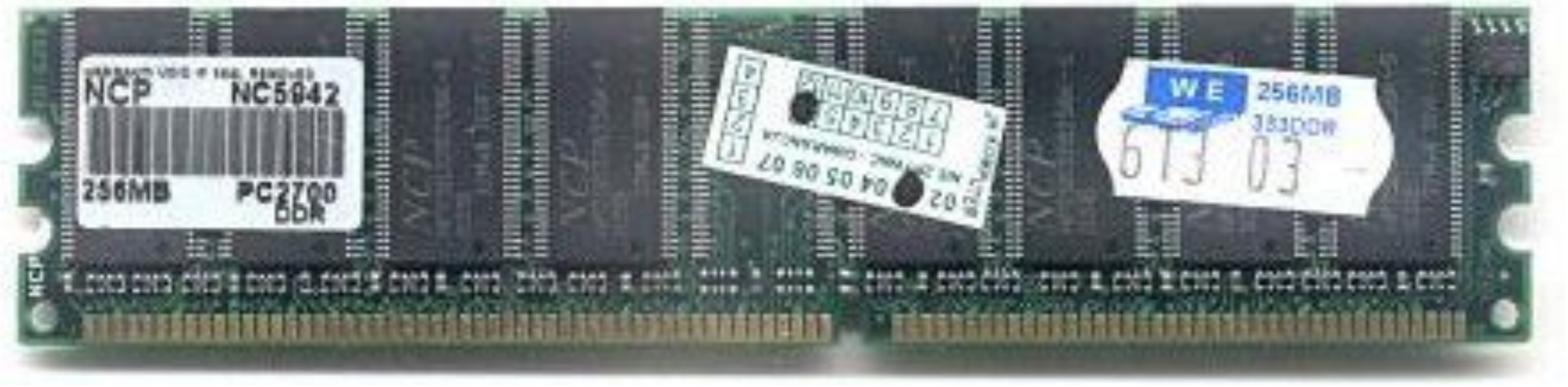
Memory Optimizations



Standard	I/O clock rate	M transfers/s	DRAM name	MiB/s/DIMM	DIMM name
DDR1	133	266	DDR266	2128	PC2100
DDR1	150	300	DDR300	2400	PC2400
DDR1	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1333	2666	DDR4-2666	21,300	PC21300

DIMM Dual Inline Memory Module





Memory Optimizations

- DDR:
  - DDR2
    - Lower power (2.5 V -> 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1333 MHz

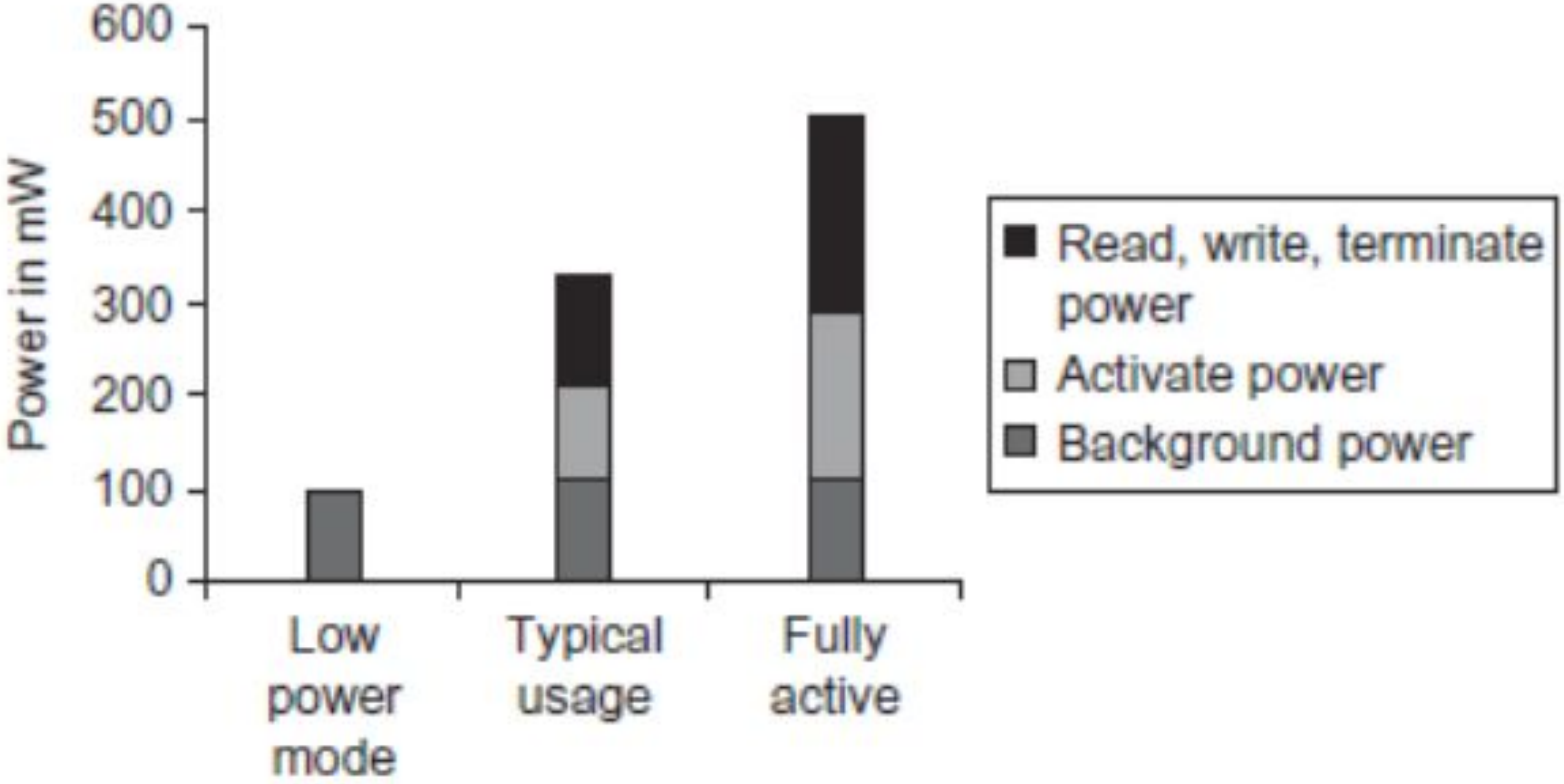
- GDDR5 is graphics memory based on DDR3

Memory Optimizations

- Reducing power in SDRAMs:
  - Lower voltage
  - Low power mode (ignores clock, continues to refresh)

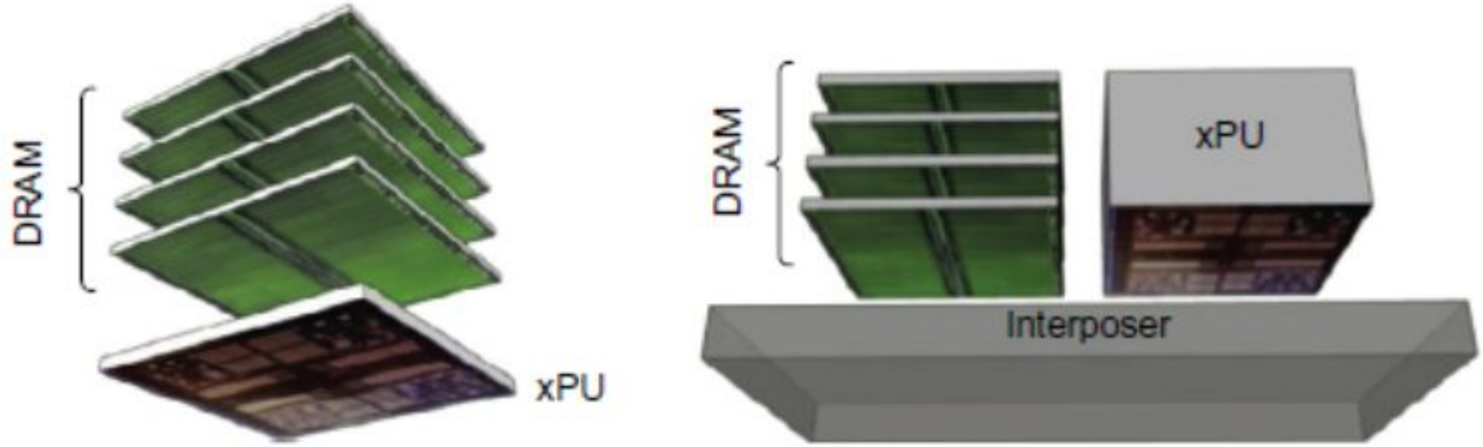
- Graphics memory:
  - Achieve 2-5 X bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering instead of socketted DIMM modules

Memory Power Consumption



Stacked/Embedded DRAMs

- Stacked DRAMs in same package as processor
  - High Bandwidth Memory (HBM)



Vertical stacking (3D)

Interposer stacking (2.5D)

Flash Memory

- Type of EEPROM
- Types: NAND (denser) and NOR (faster)
- NAND Flash:
  - Reads are sequential, reads entire page (.5 to 4 KiB)
  - 25 us for first byte, 40 MiB/s for subsequent bytes
  - SDRAM: 40 ns for first byte, 4.8 GB/s for subsequent bytes
  - 2 KiB transfer: 75 uS vs 500 ns for SDRAM, 150X slower
  - 300 to 500X faster than magnetic disk

NAND Flash Memory

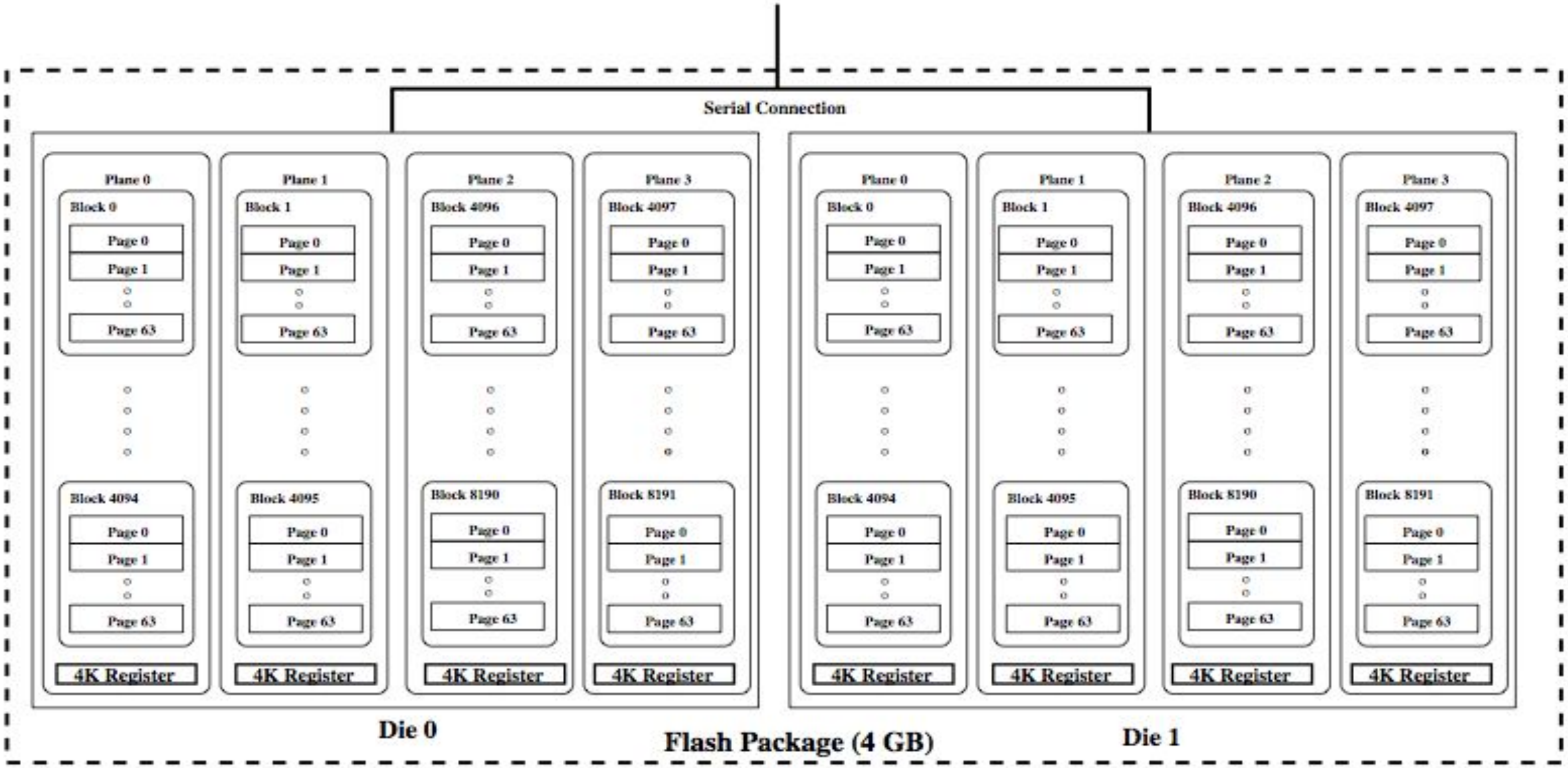
- Must be erased (in blocks) before being overwritten
- Nonvolatile, can use as little as zero power
- Limited number of write cycles (~100,000)
- $2/GiB, compared to $20-40/GiB for SDRAM and $0.09 GiB for magnetic disk

- Phase-Change/Memrister Memory
  - Possibly 10X improvement in write performance and 2X improvement in read performance

Solid State Drives

NAND Flash Memory

- Main storage component of Solid State Drive (SSD)
  - USB Drive, cell phone, touch pad…



NAND Flash Memory

- Advantages of NAND flash
  - Fast random read (25 us)
  - Energy efficiency
  - High reliability (no moving parts) compared to harddisks
- Widely deployed in high-end laptops
  - Macbook air, ThinkPad X series, touch pad…
- Increasingly deployed in enterprise environment either as a secondary cache or main storage

NAND Flash Memory

- Disadvantages of SSD
  - Garbage collection (GC) problem of SSD
    - Stemmed from the *out-of-place* update characteristics
    - Update requests invalidate old version of pages and then write new version of these pages to a new place
    - Copy valid data to somewhere else (increasing number of IOs)
    - Garbage collection is periodically started to erase victim blocks and copy valid pages to the free blocks (slow erase: 10xW,100xR)
  - Blocks in the SSD have a limited number of erase cycles
    - 100,000 for Single Level Chip (SLC), 5,000-10,000 for Multiple Level Chip (MLC), can be as low as 3,000
    - May be quickly worn out in enterprise environment
  - Performance is very unpredictable
    - Due to unpredictable triggering of the time-consuming GC process

Hybrid Main Memory System

- DRAM + Flash Memory
  - Uses small DRAM as a cache to buffer writes and cache reads by leveraging access locality
  - Uses large flash memory to store cold data
  - Advantages
    - Similar performance as DRAM
    - Low power consumption
    - Low costs

Comparison SSD - HDD

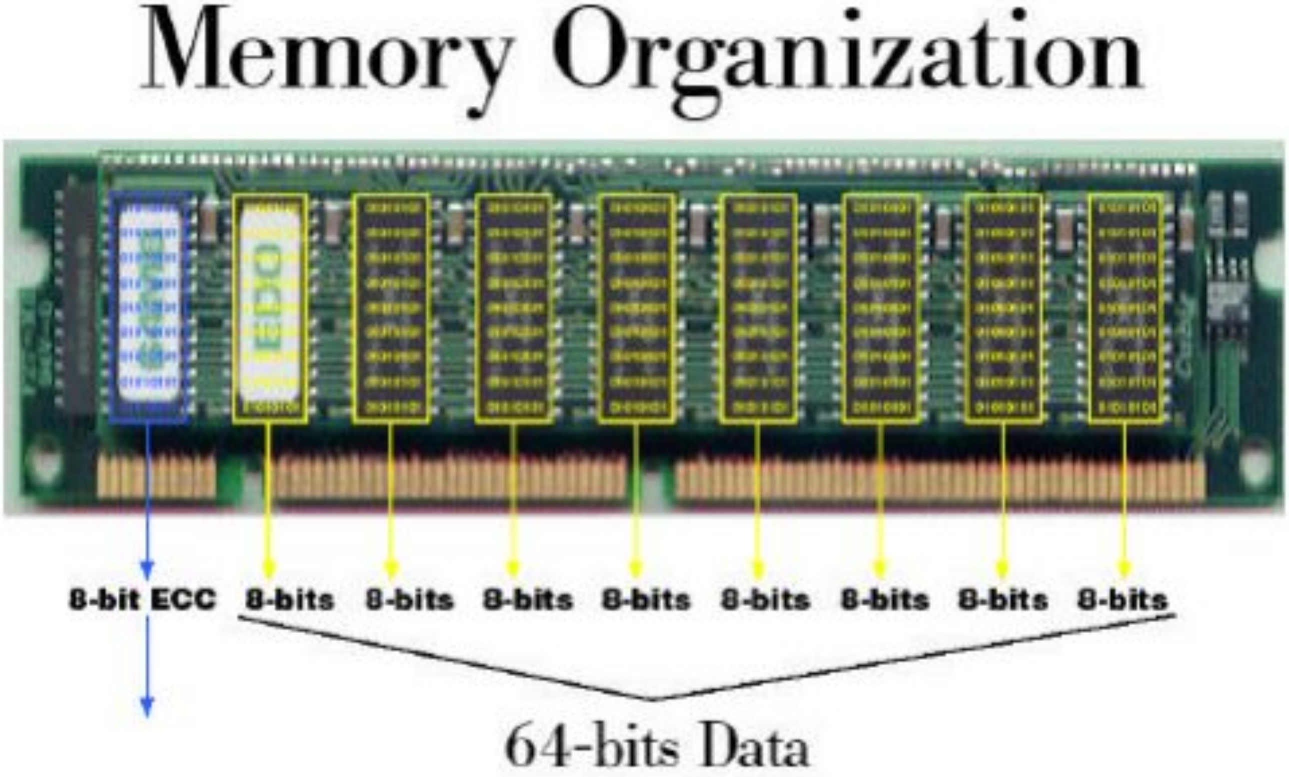
Attribute	SSD	HDD
Random access time	0.1 ms	5-10 ms
Bandwidth	100-500 MB/s	100 MB/s sequential
Price/GB	0.9-2	0.1$
Size	Up to 2TB, 250GB common	4TB
Power consumption	5 watts	Up to 20 watts
Read/write symmetry	No	Yes
Noise	No	Yes (spin, rotate)

Memory Dependability

- Memory is susceptible to cosmic rays
- *Soft errors*: dynamic errors
  - Detected and fixed by error correcting codes (ECC)
- *Hard errors*: permanent errors
  - Use spare rows to replace defective rows

- Chipkill: a RAID-like error recovery technique

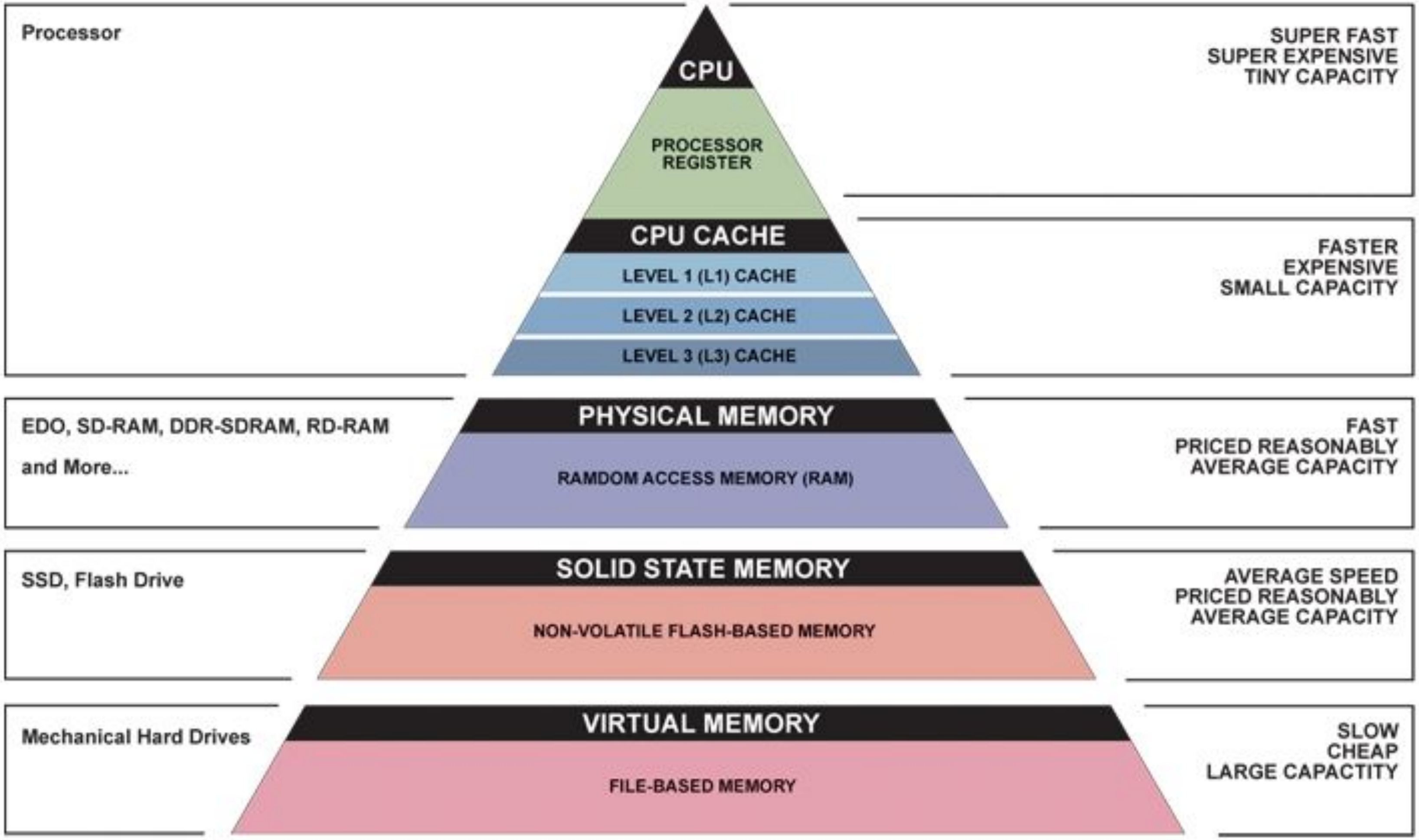
Memory Dependability



Memory Dependability

- A Redundant Array of Inexpensive DRAM (RAID) processor chip is directly placed on the memory DIMM.
- The RAID chip calculates an ECC checksum for the contents of the entire set of chips for each memory access and stores the result in extra memory space on the protected DIMM.
- Thus, when a memory chip on the DIMM fails, the RAID result can be used to "back up" the lost data.

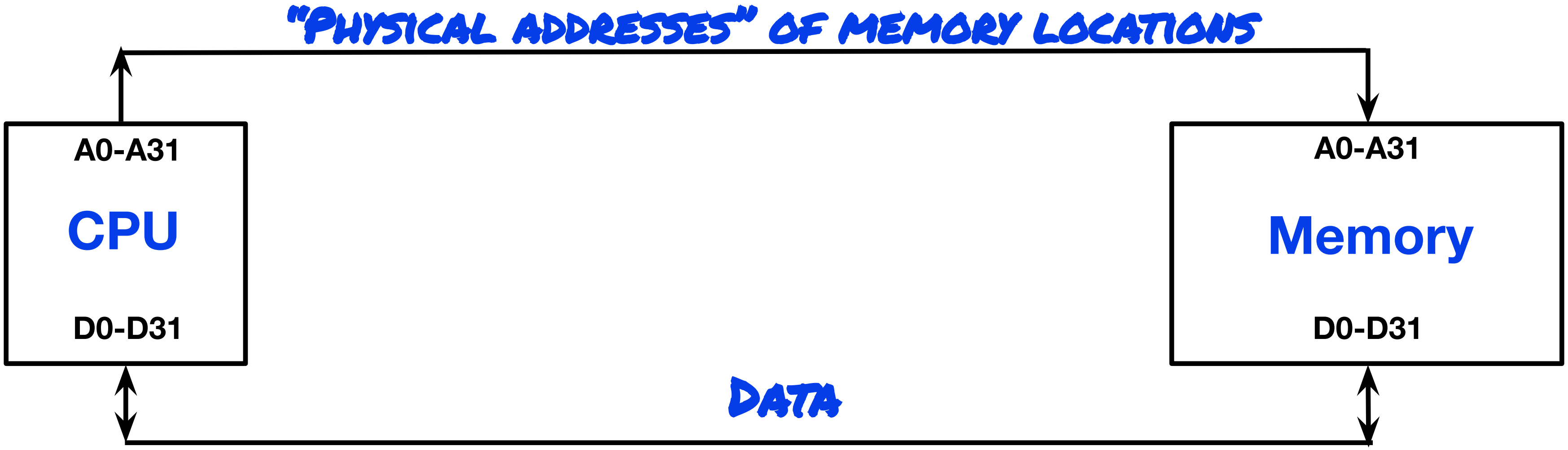
Computer Memory Hierarchy



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

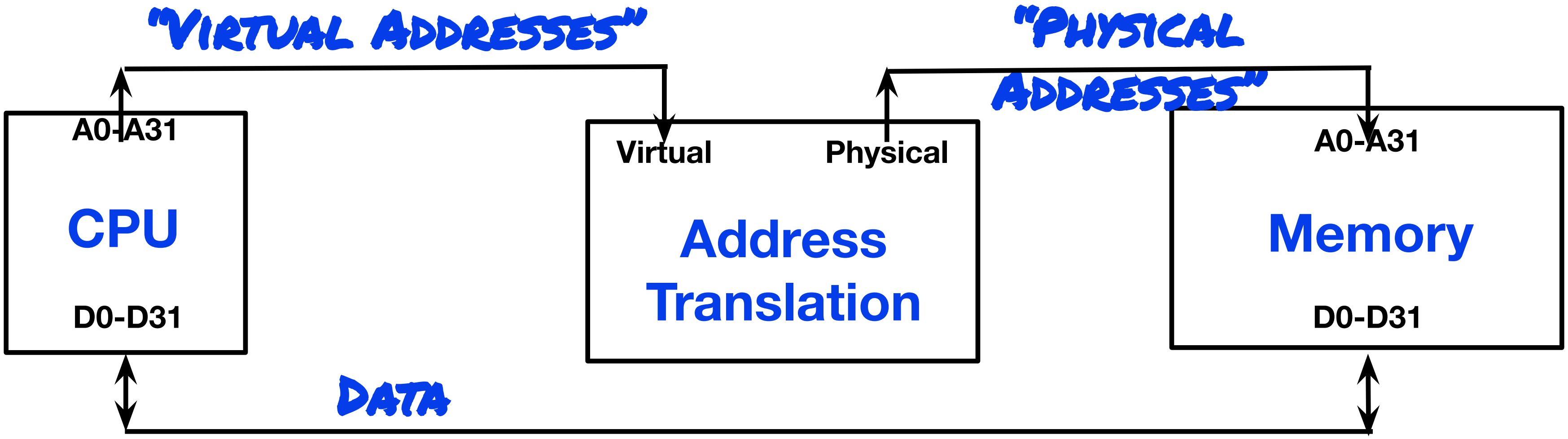
http://www.bit-tech.net/hardware/memory/2007/11/15/the_secrets_of_pc_memory_part_1/3

The Limits of Physical Addressing



- All programs share one address space: The physical address space
- Machine language programs must be aware of the machine organization
- No way to prevent a program from accessing any machine resource

Solution: Add a Layer of Indirection

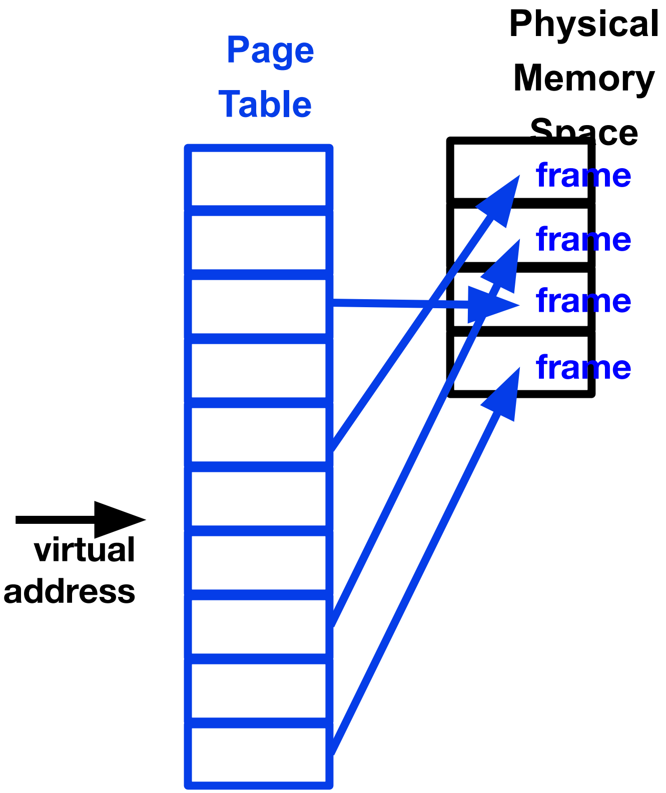


- User programs run in a standardized virtual address space
- Address Translation hardware, managed by the operating system (OS), maps virtual address to physical memory
- Hardware supports "modern" OS features: Protection, Translation, Sharing

Three Advantages of Virtual Memory

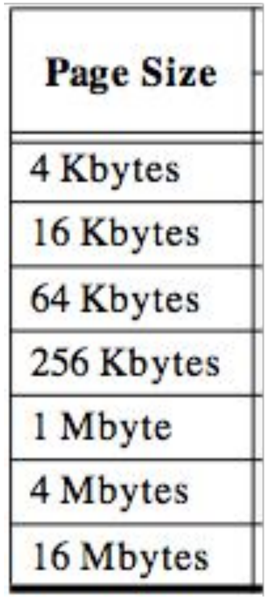
- Translation:
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot!)
  - Only the most important part of program ("Working Set") must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.
- Protection:
  - Different threads (or processes) protected from each other.
  - Different pages can be given special behavior
    - (Read Only, Invisible to user programs, etc).
  - Kernel data protected from User programs
  - Very important for protection from malicious programs
- Sharing:
  - Can map same physical page to multiple users ("Shared memory")

Page tables encode virtual address spaces



A virtual address space
is divided into blocks
of memory called pages

A machine usually
supports
pages of a few
sizes
(MIPS R4000):



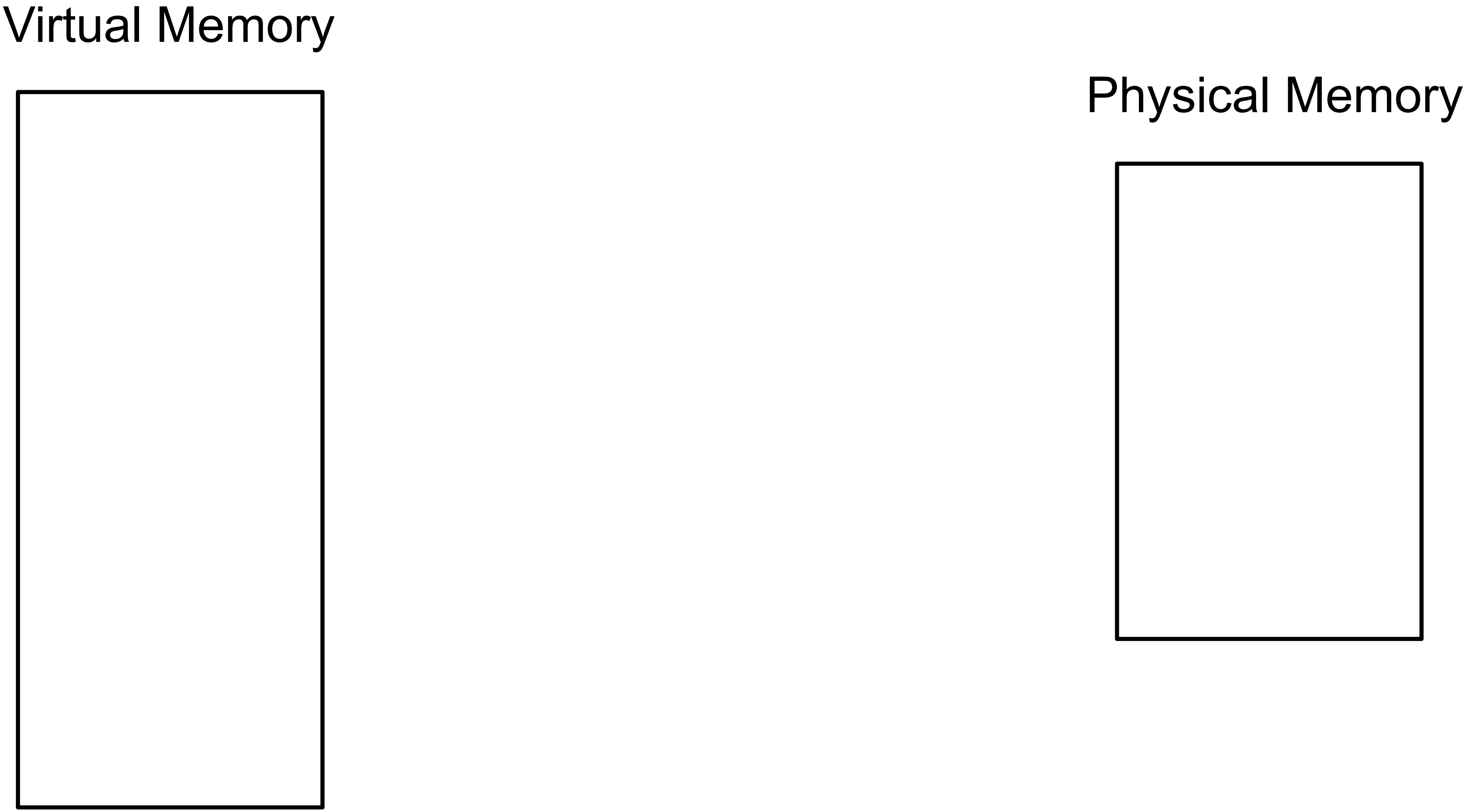
Page Size
4 Kbytes
16 Kbytes
64 Kbytes
256 Kbytes
1 Mbyte
4 Mbytes
16 Mbytes

A page table is indexed by a
virtual address

OS MANAGES
THE PAGE
TABLE FOR
EACH ASID
(ADDR. SPACE
ID)

A valid page table entry codes physical memory
"frame" address for the page

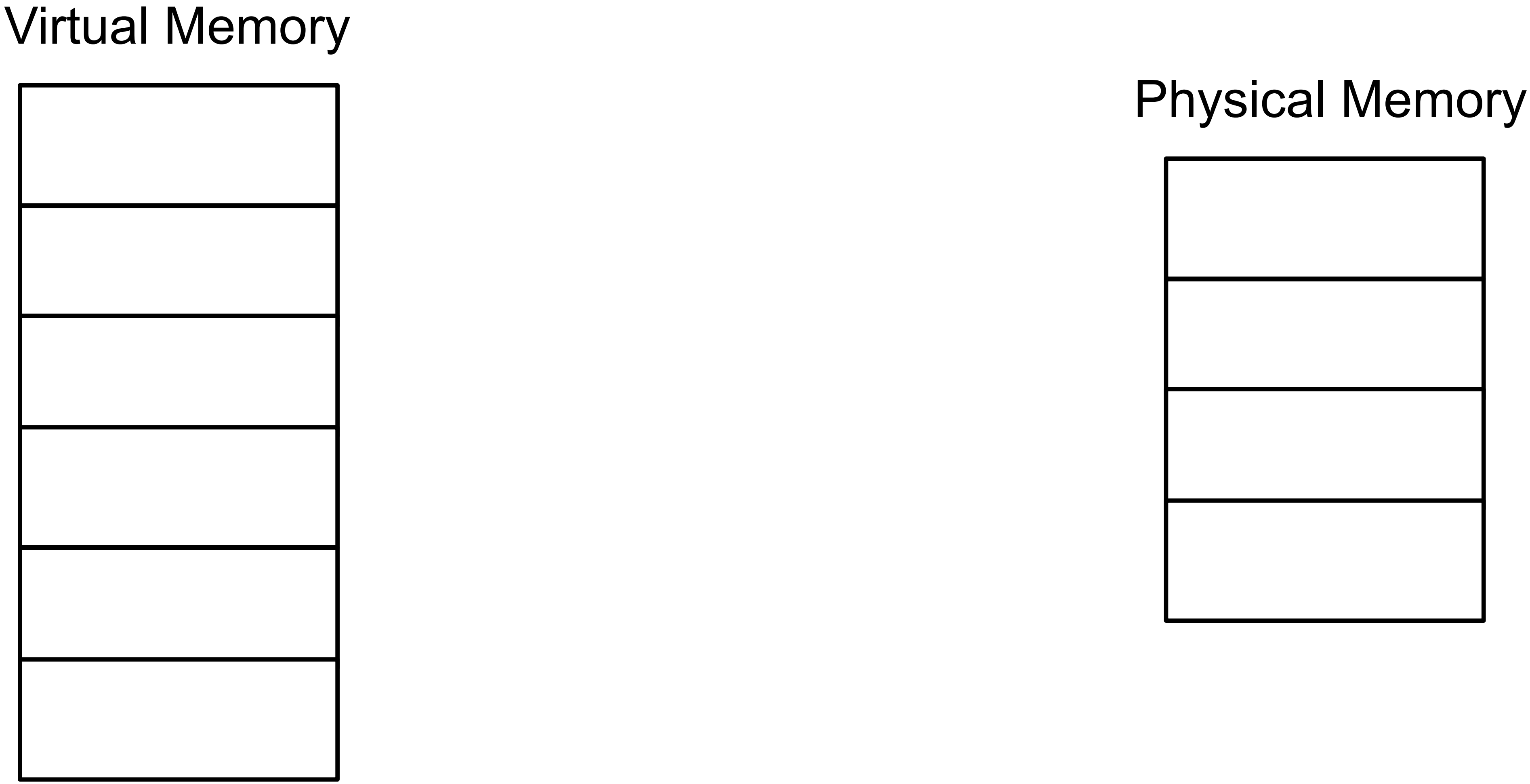
An Example of Page Table



Virtual Memory

Physical Memory

Dividing the address space by a page size

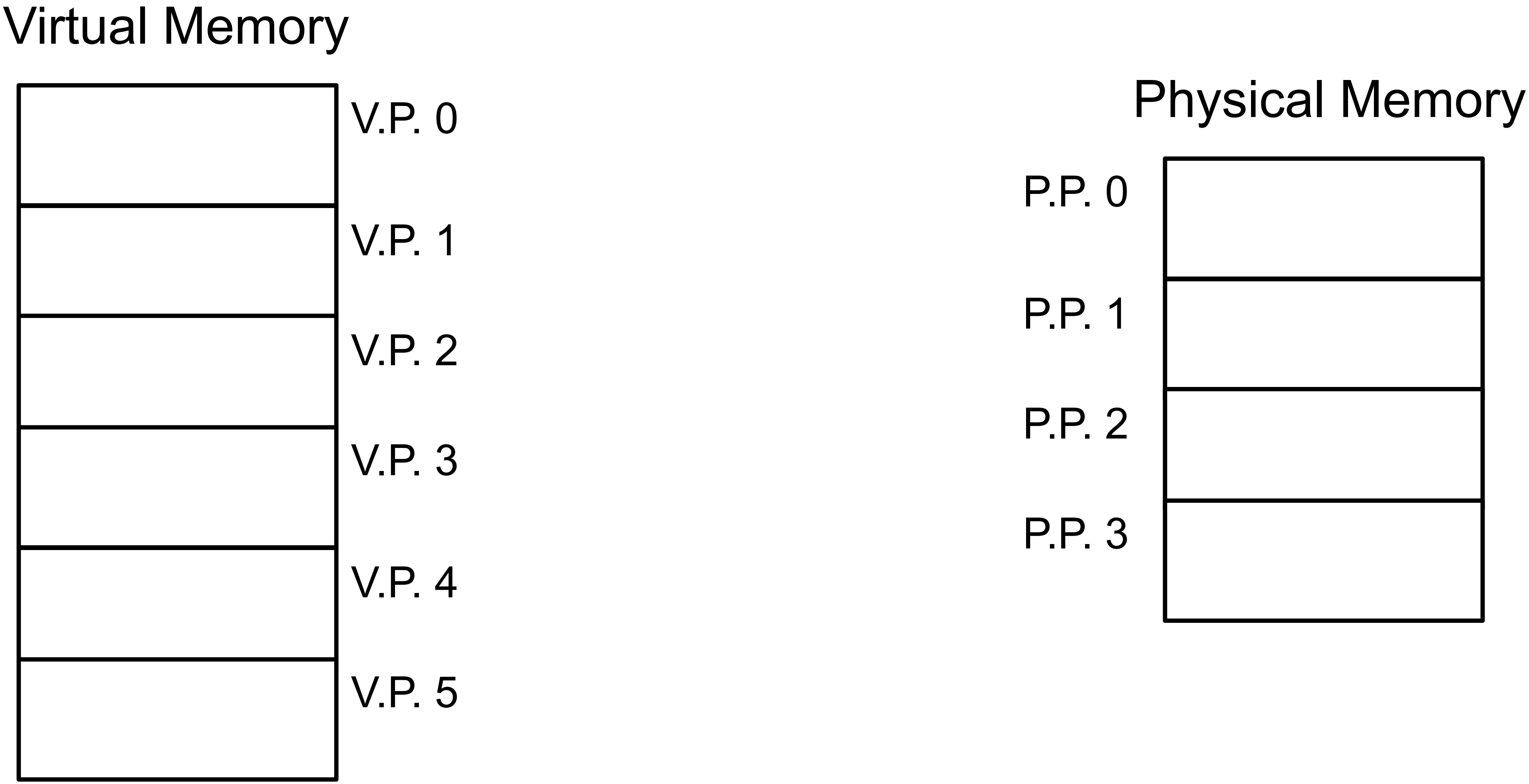


Virtual Memory

Physical Memory

Page size:4KB

Virtual Page & Physical Page



Virtual Memory

V.P. 0

V.P. 1

V.P. 2

V.P. 3

V.P. 4

V.P. 5

Physical Memory

P.P. 0

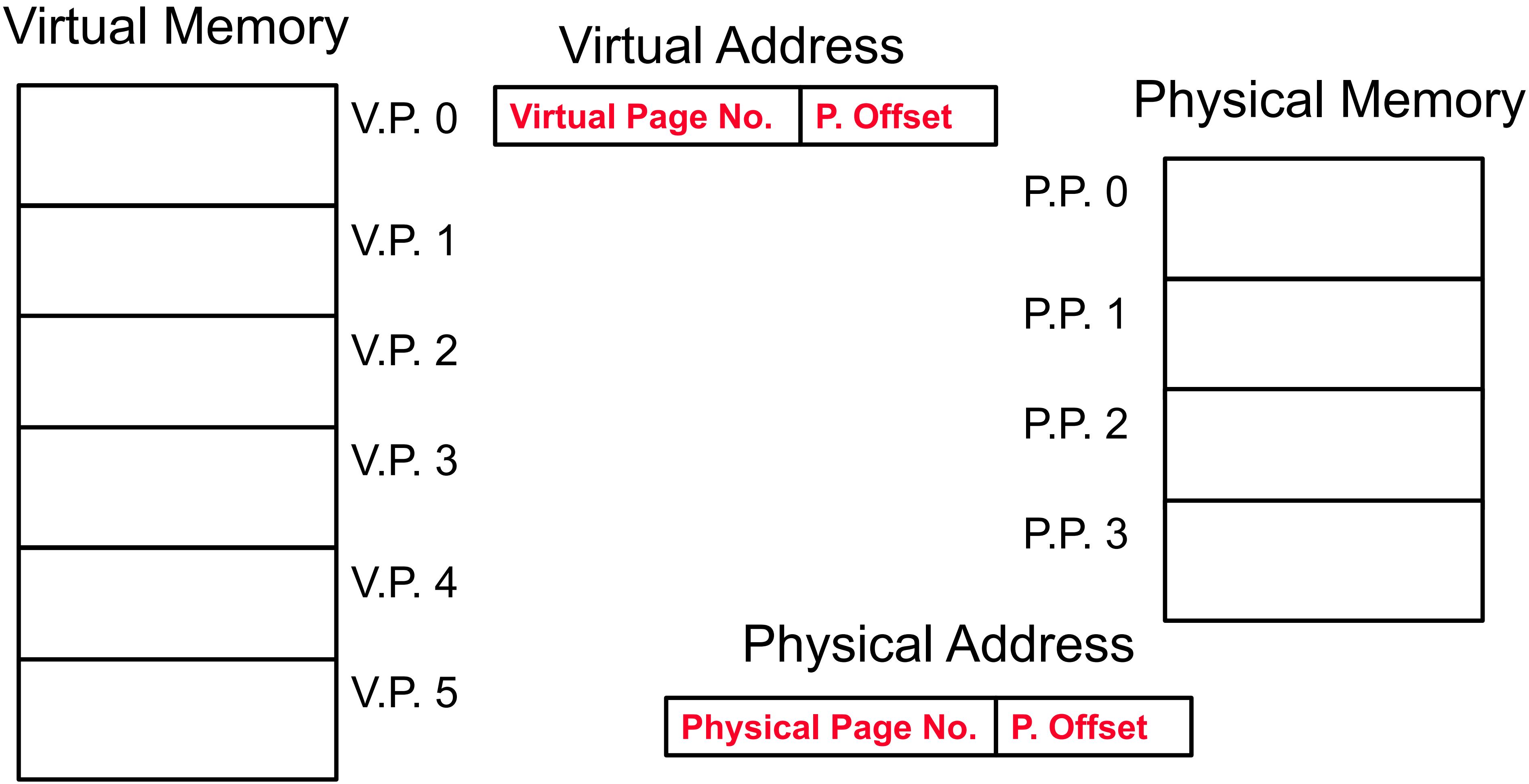
P.P. 1

P.P. 2

P.P. 3

Page size:4KB

Addressing



Virtual Memory

V.P. 0

V.P. 1

V.P. 2

V.P. 3

V.P. 4

V.P. 5

Virtual Address

Virtual Page No.	P. Offset

Physical Memory

P.P. 0

P.P. 1

P.P. 2

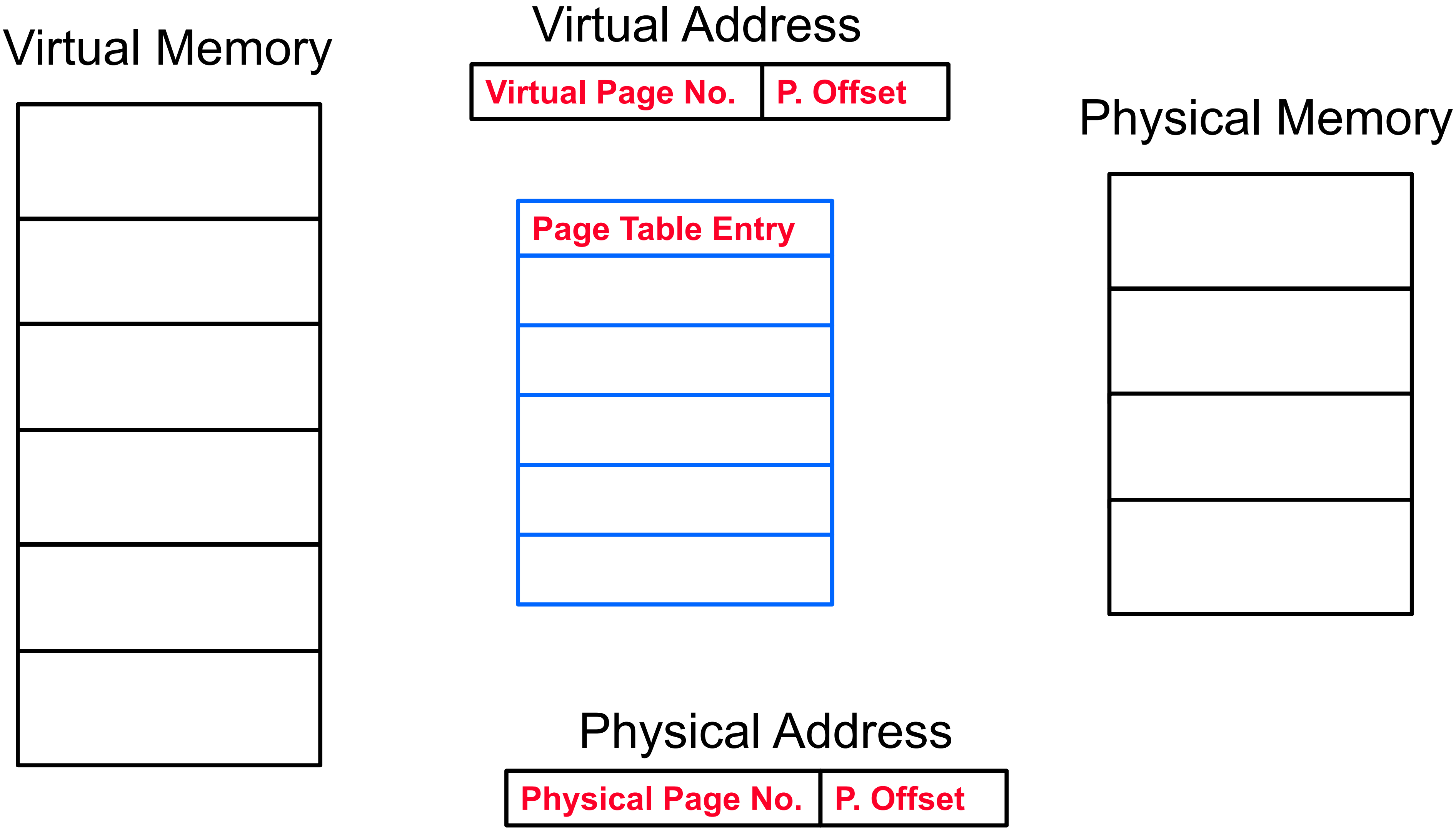
P.P. 3

Physical Address

Physical Page No.	P. Offset

Page size:4KB

Addressing



Virtual Memory

Virtual Address

Virtual Page No. | P. Offset

Page Table Entry

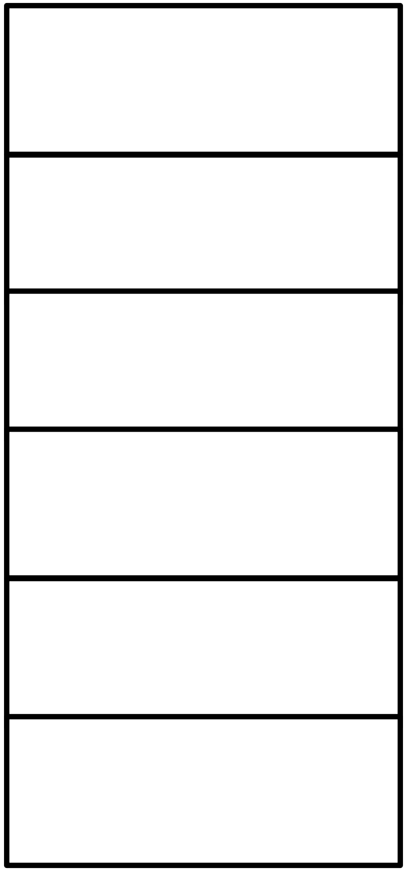
Physical Memory

Physical Address

Physical Page No. | P. Offset

Addressing

Virtual Memory



Virtual Address

Virtual Page No.	P. Offset

Page Table Entry

V	P	R	D	Physical Page No.

Valid/Present Bit

If set, page being pointed is resident in memory

Otherwise, on disk or not allocated

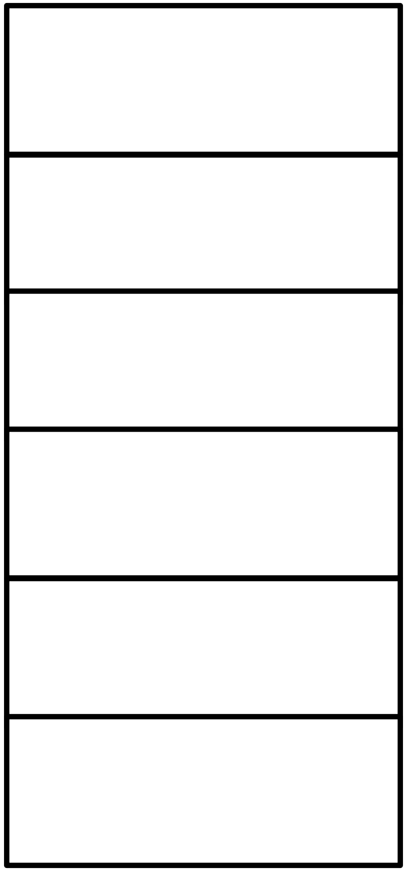
Physical Address

Physical Page No.	P. Offset

Physical Memory

Addressing

Virtual Memory



Virtual Address

Virtual Page No.	P. Offset

Page Table Entry

V	P	R	D	Physical Page No.

Physical Memory

Protection Bits

Restrict access;

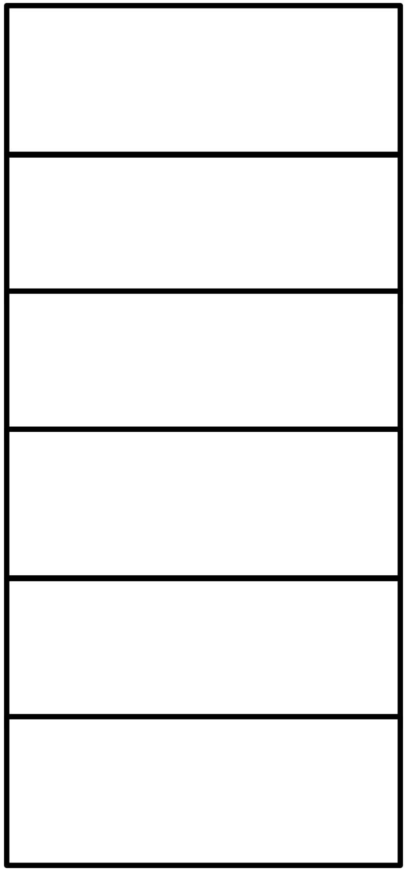
read-only, read/write,
system-only access

Physical Address

Physical Page No.	P. Offset

Addressing

Virtual Memory



Virtual Address

Virtual Page No.	**P. Offset**

Page Table Entry

V	**P**	**R**	**D**	**Physical Page No.**

Reference Bit

Needed by replacement policies

If set, page has been referenced

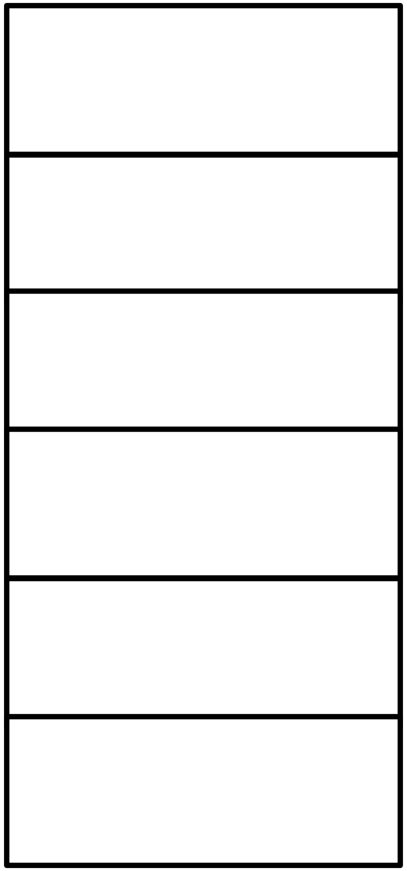
Physical Address

Physical Page No.	**P. Offset**

Physical Memory

Page Table Entry

Virtual Memory



Virtual Address

Virtual Page No.	P. Offset

Page Table Entry

V	P	R	D	Physical Page No.

Dirty Bit

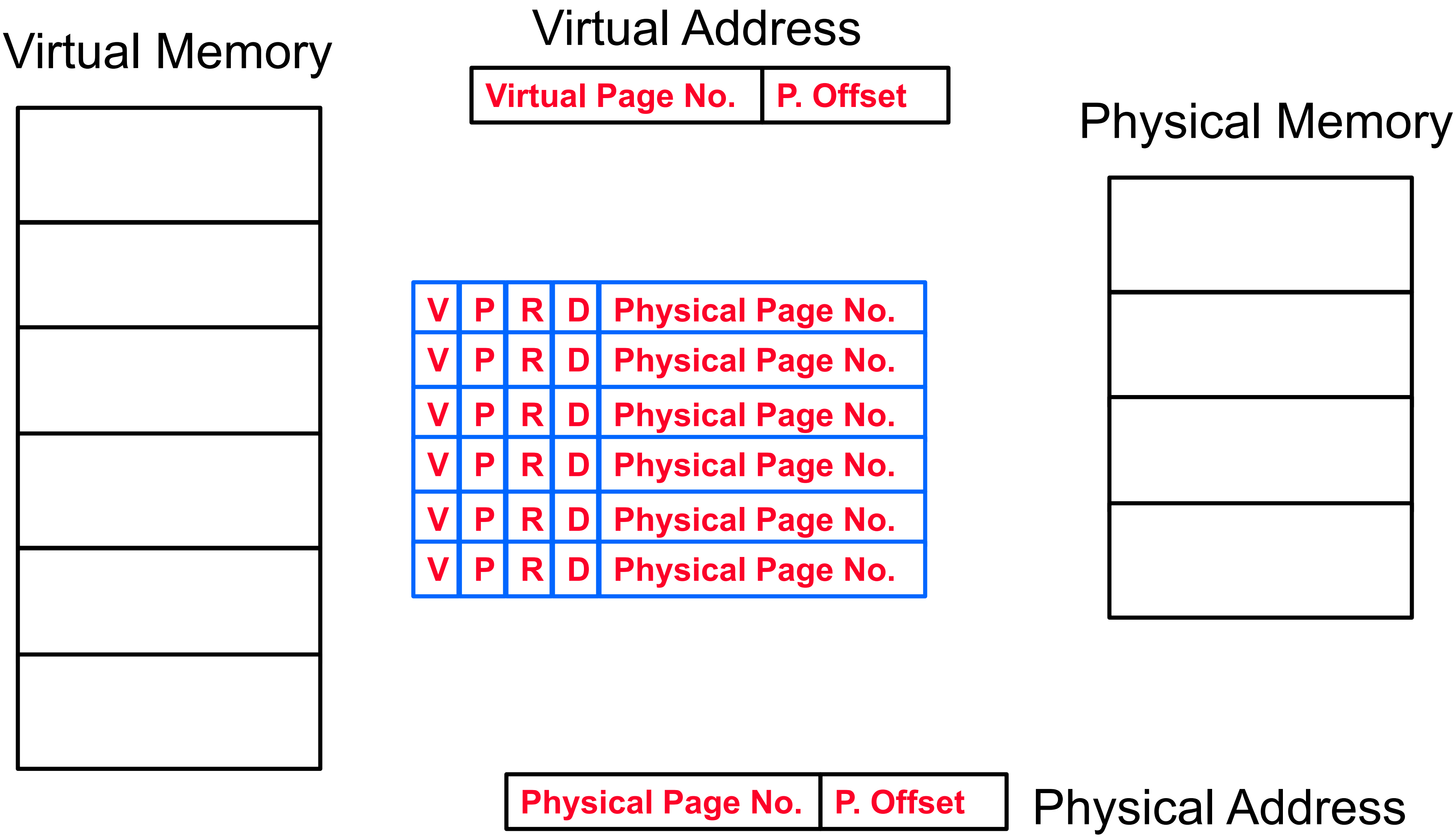
If set, at least one word in page has been modified

Physical Address

Physical Page No.	P. Offset

Physical Memory

Page Table Entry



Virtual Memory

Virtual Address

Virtual Page No.	P. Offset

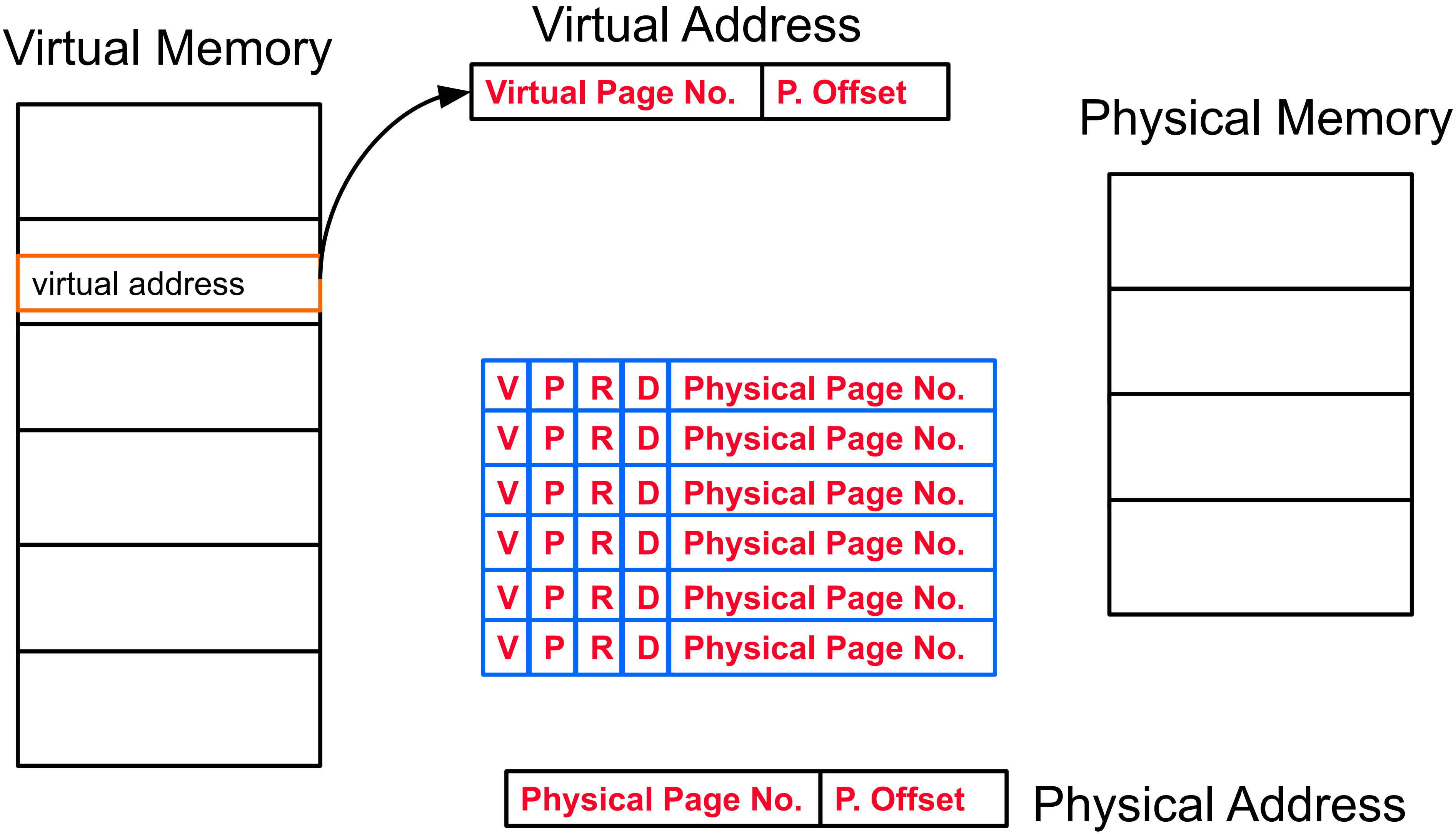
Physical Memory

V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.

Physical Page No.	P. Offset

Physical Address

Page Table Lookup



Virtual Memory

virtual address

Virtual Address

Virtual Page No.	P. Offset

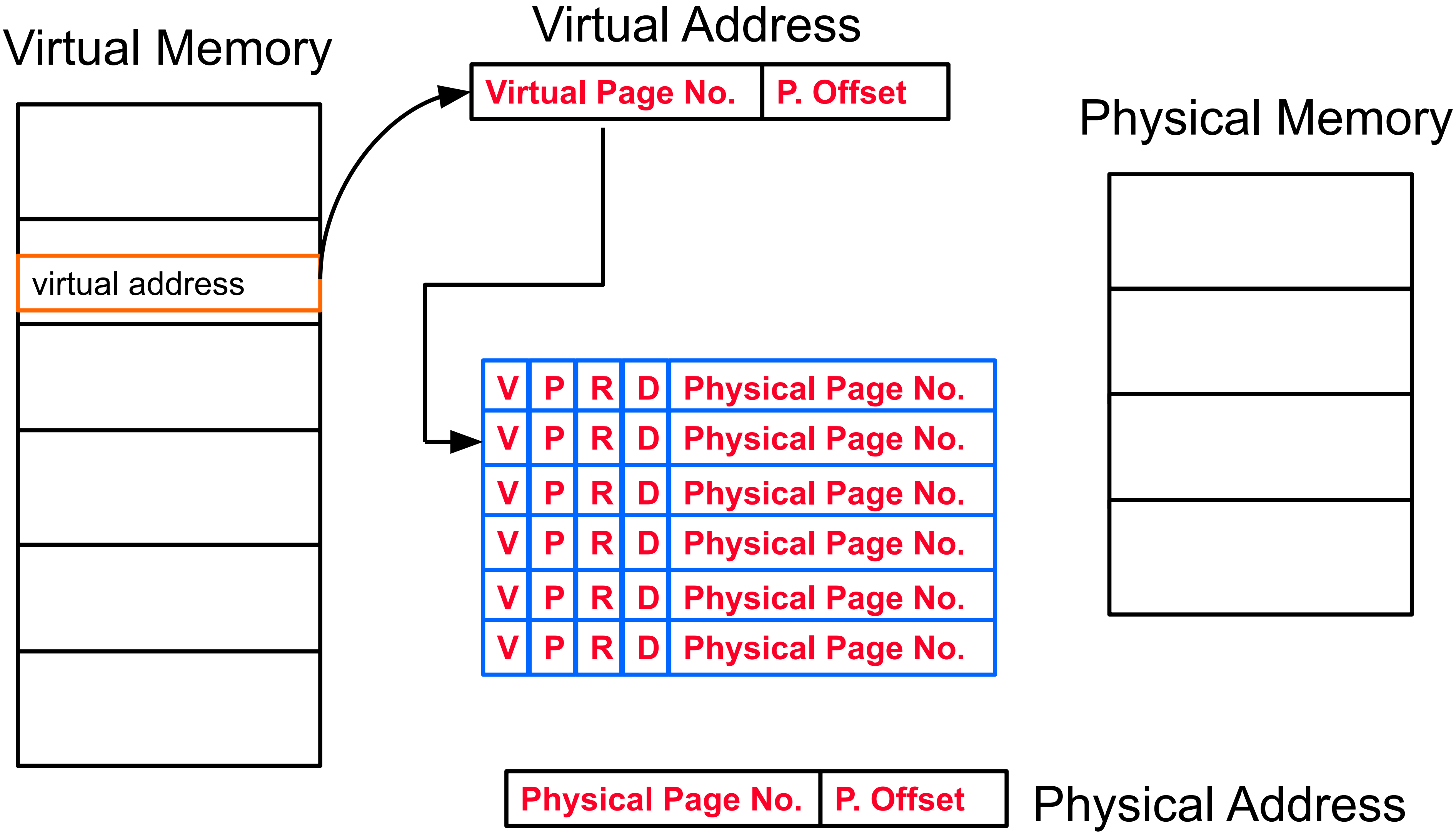
Physical Memory

V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.

Physical Page No.	P. Offset

Physical Address

Page Table Lookup



Virtual Memory

virtual address

Virtual Address

Virtual Page No.	P. Offset

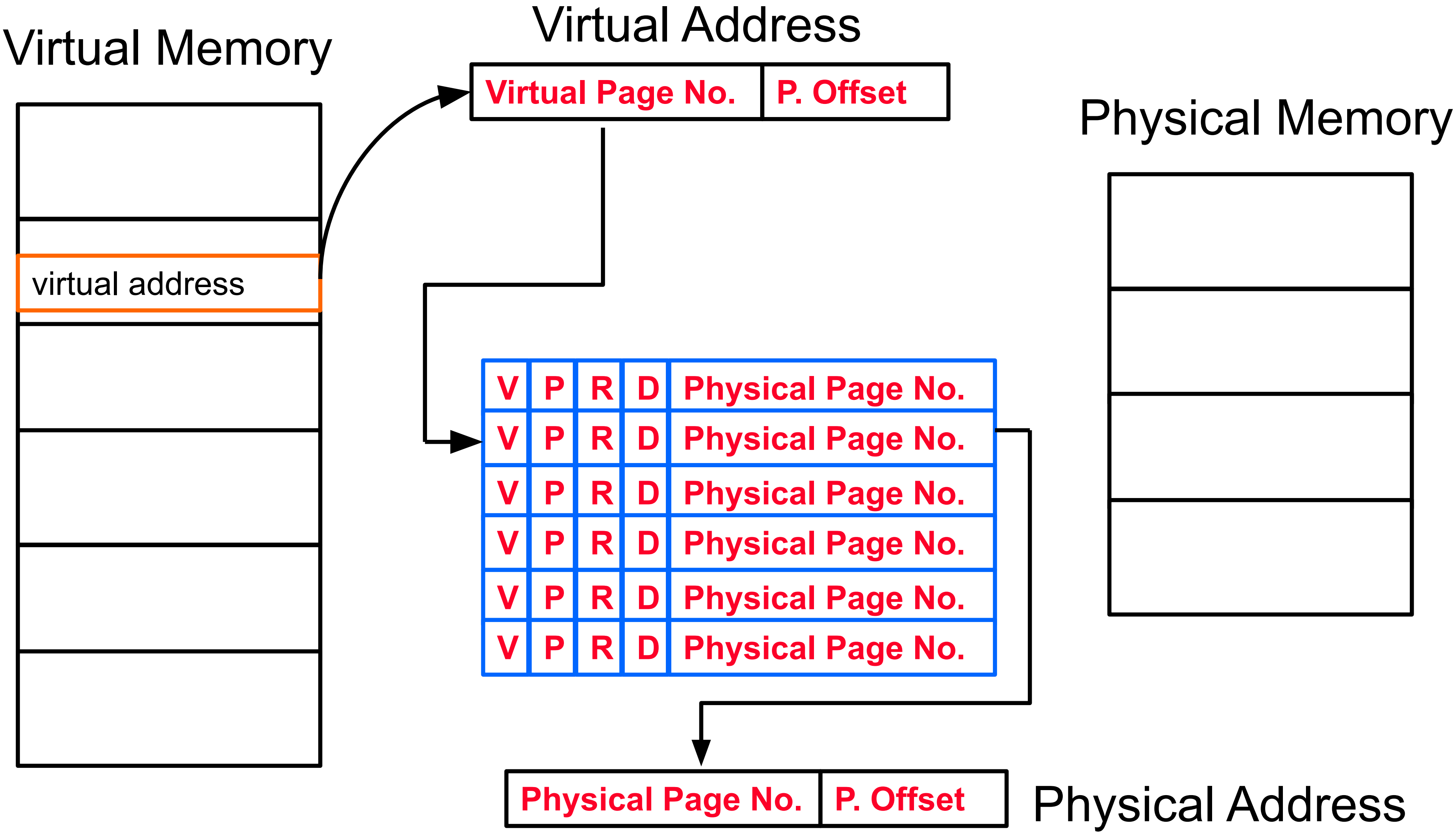
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.

Physical Memory

Physical Page No.	P. Offset

Physical Address

Page Table Lookup



Virtual Memory

virtual address

Virtual Address

Virtual Page No. | P. Offset

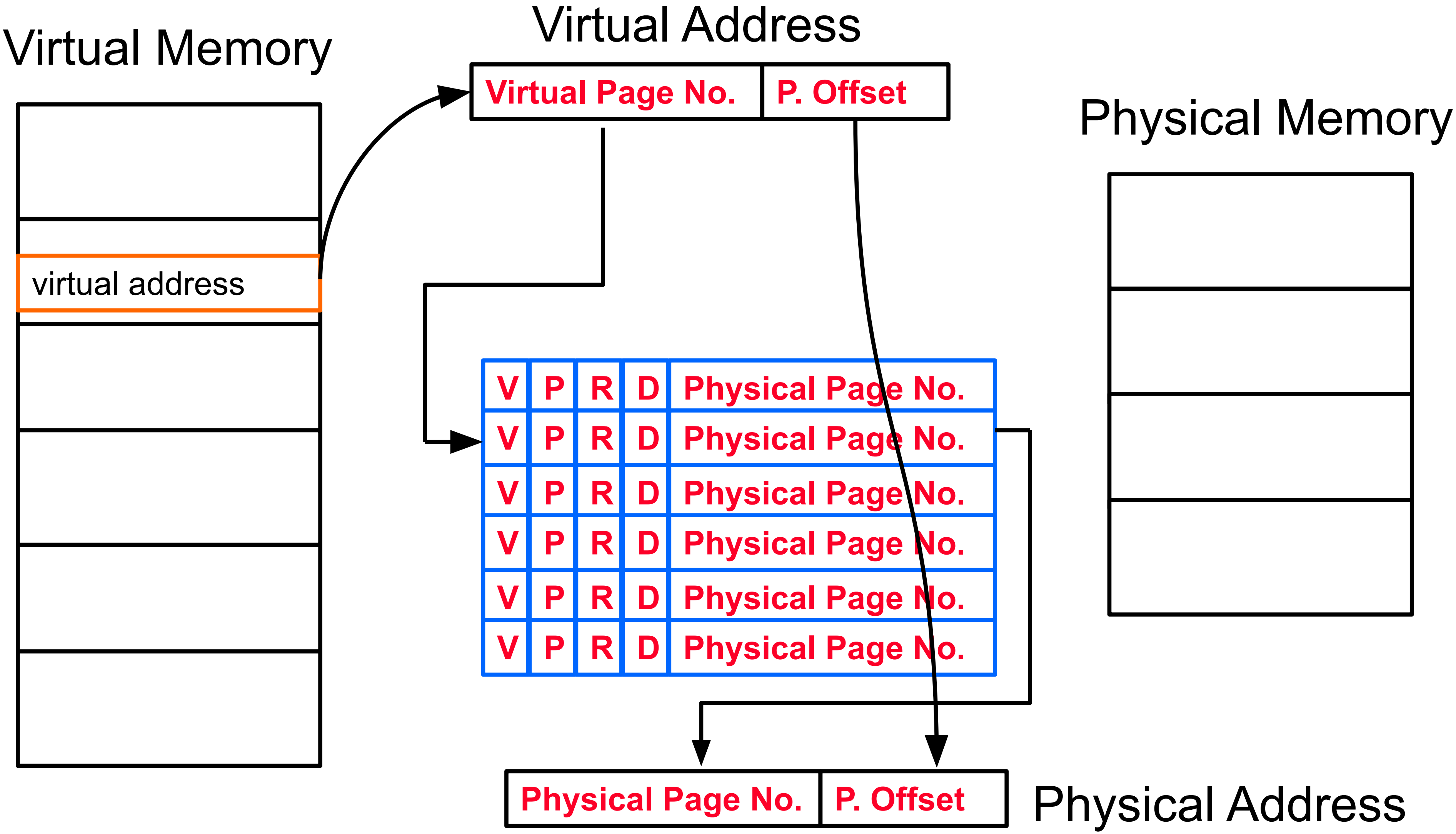
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.

Physical Memory

Physical Page No. | P. Offset

Physical Address

Page Table Lookup



Virtual Memory

virtual address

Virtual Address

Virtual Page No.	P. Offset

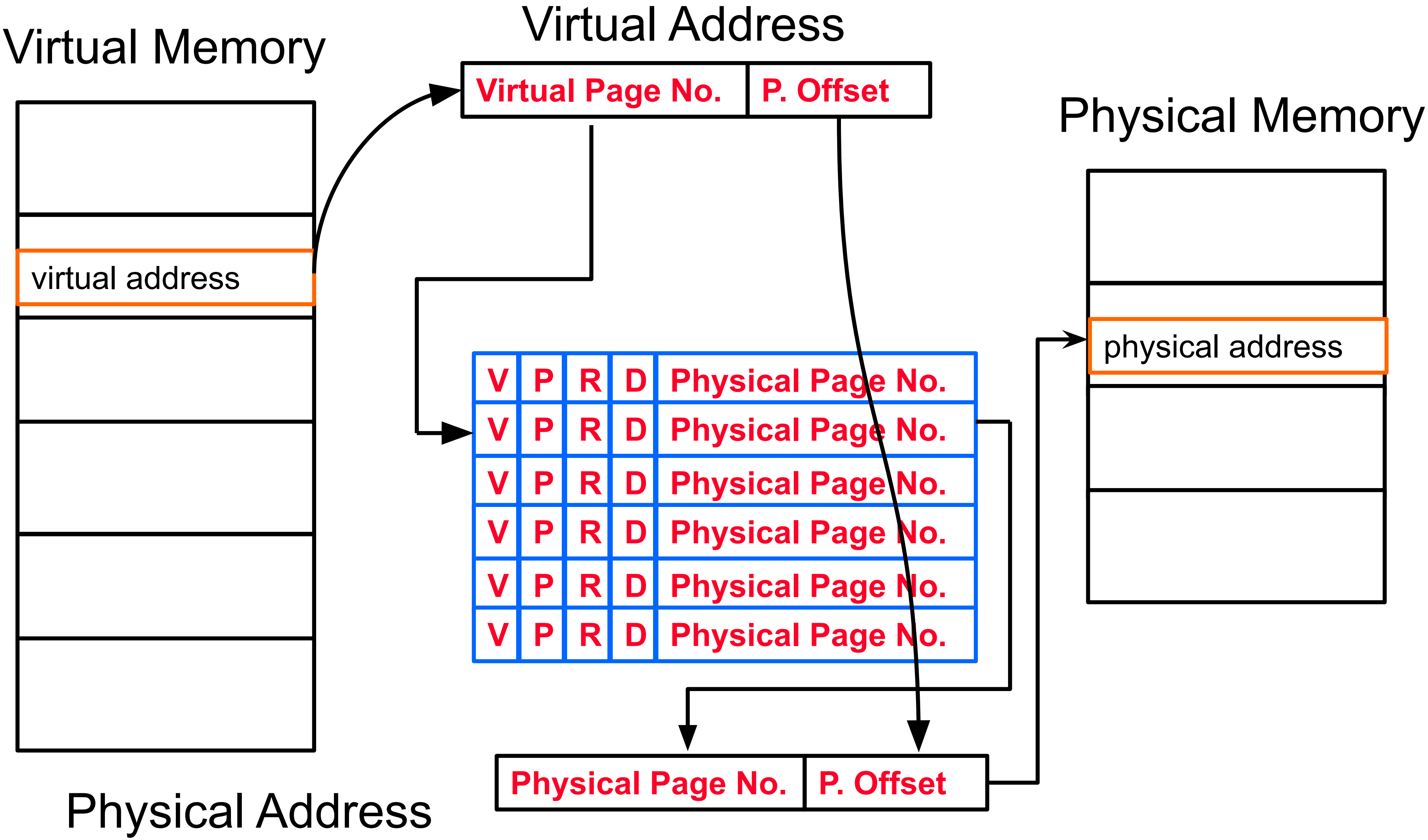
Physical Memory

V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.

Physical Page No.	P. Offset

Physical Address

Page Table Lookup



Virtual Memory

virtual address

Virtual Address

Virtual Page No.	P. Offset

V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.

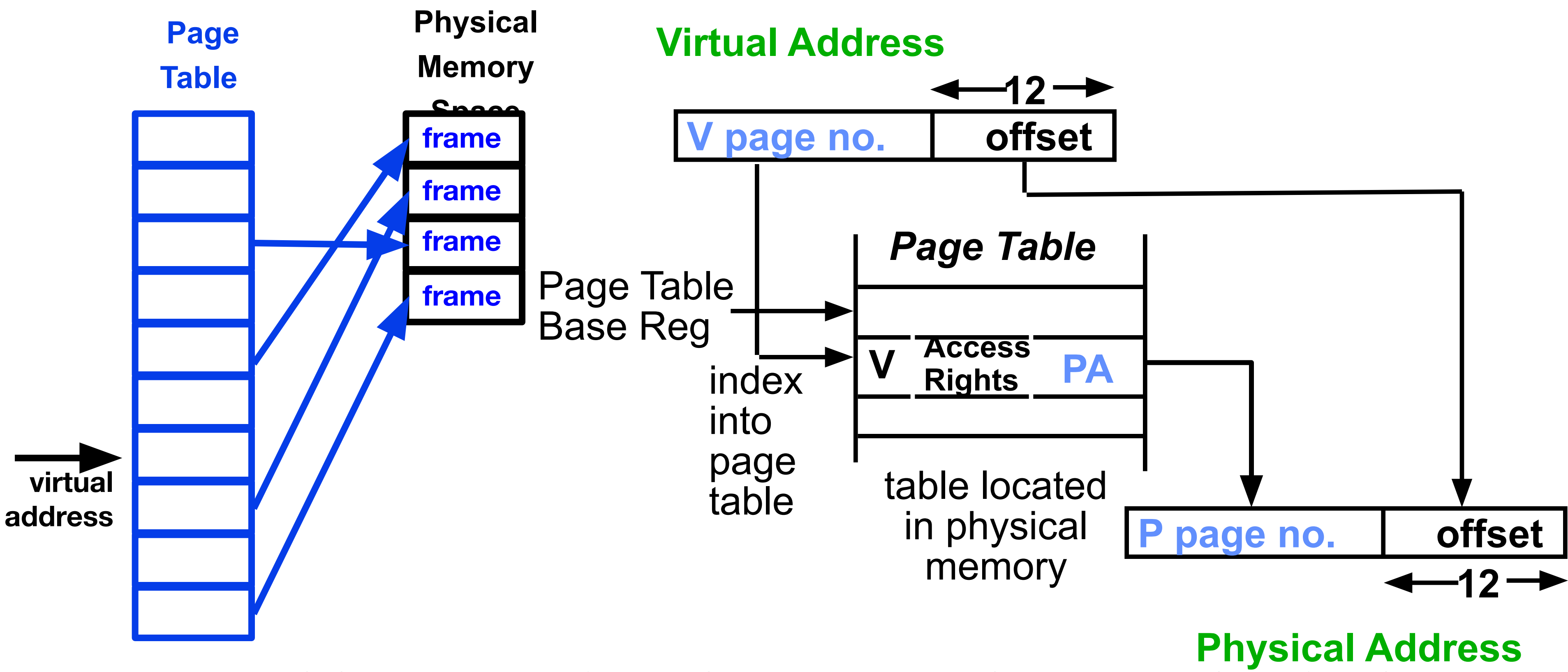
Physical Memory

physical address

Physical Address

Physical Page No.	P. Offset

Details of Page Table



- Page table maps virtual page numbers to physical frames ("PTE" = Page Table Entry)
- Virtual memory => treat memory ≈ cache for disk
- 4 fundamental questions: placement, identification, replacement, and write policy?

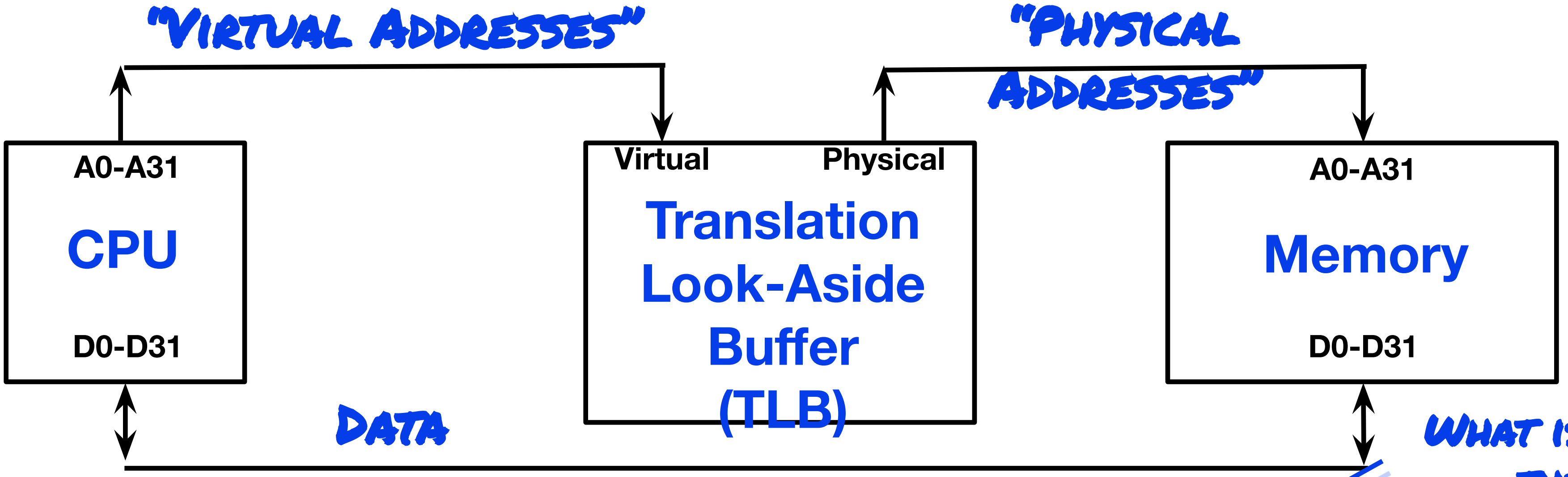
4 Fundamental Questions

- Placement
  - Operating systems allow blocks to be placed anywhere in main memory
- Identification
  - Page Table, Inverted Page Table
- Replacement
  - Almost all operating systems try to use LRU
- Write Policies
  - Always write back

Latency

- Since Page Table is located in main memory, it takes one memory access latency to finish an address translation;
- As a result, a load/store operation from/to main memory needs two memory access latency in total;
- Considering the expensive memory access latency, the overhead of page table lookup should be optimized;
- How?
  - Principle of Locality
  - Caching

MIPS Address Translation: How does it work?



"Virtual Addresses"

"Physical Addresses"

A0-A31

CPU

D0-D31

Virtual

Physical

Translation Look-Aside Buffer (TLB)

A0-A31

Memory

D0-D31

Data

What is the table of mappings that it caches?

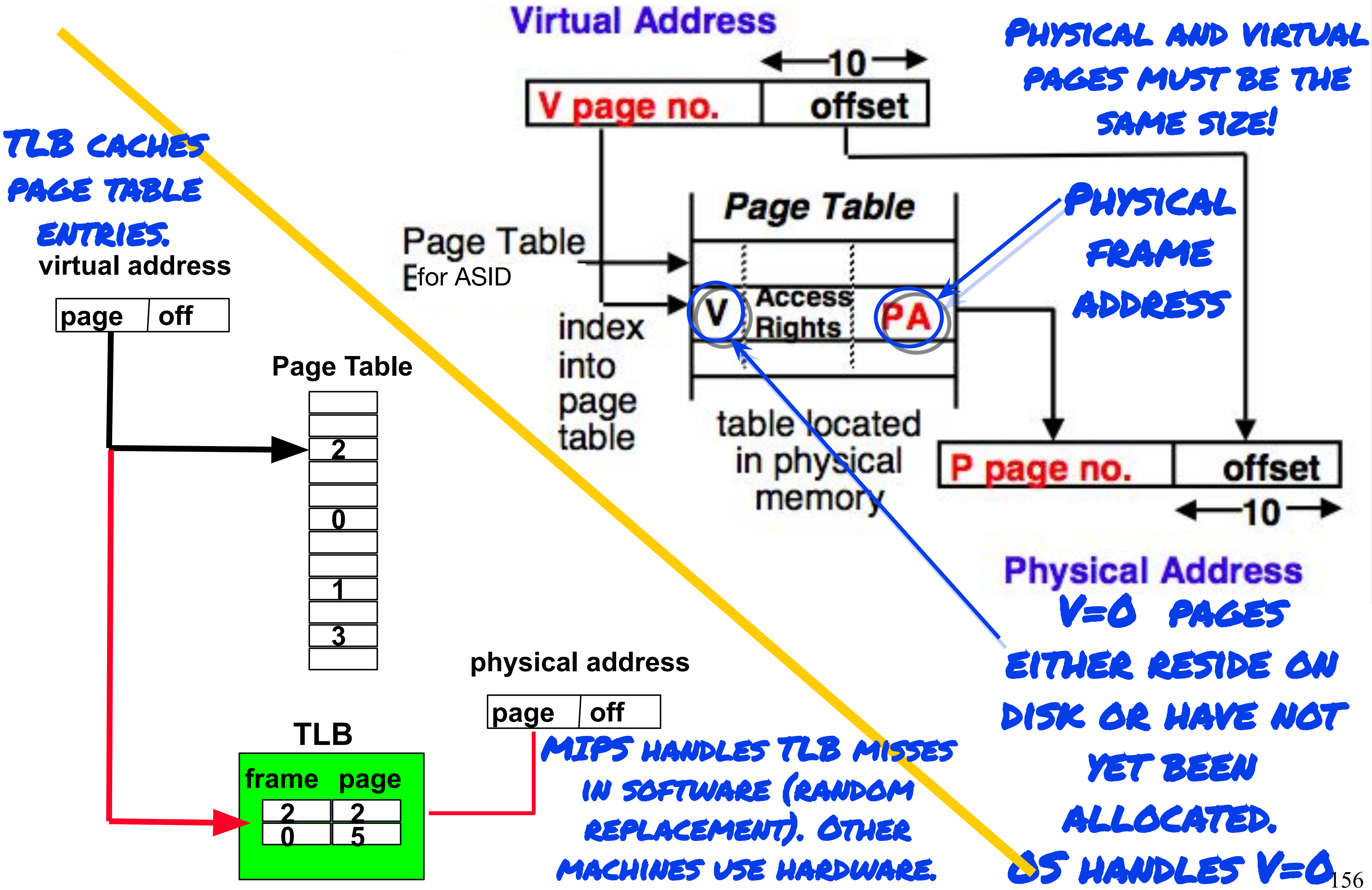
Translation Look-Aside Buffer (TLB)

A small fully-associative cache of mappings from virtual to physical addresses

TLB also contains protection bits for virtual address

Fast common case: Virtual address is in TLB, process has permission to read/write it.

The TLB caches page table entries



TLB CACHES PAGE TABLE ENTRIES.

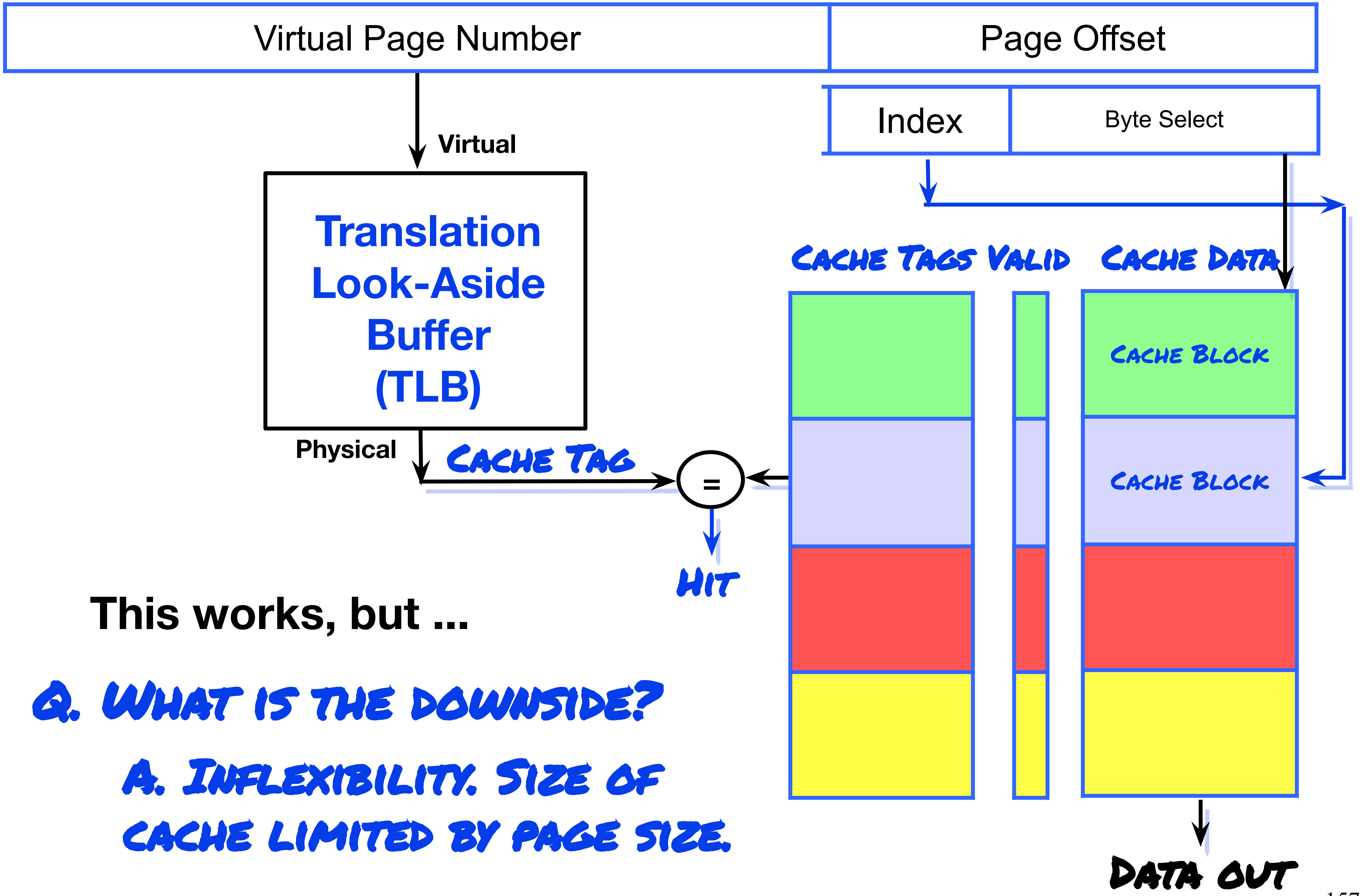
PHYSICAL AND VIRTUAL PAGES MUST BE THE SAME SIZE!

PHYSICAL FRAME ADDRESS

V=0 PAGES EITHER RESIDE ON DISK OR HAVE NOT YET BEEN ALLOCATED. OS HANDLES V=0 "PAGE FAULT"

MIPS HANDLES TLB MISSES IN SOFTWARE (RANDOM REPLACEMENT). OTHER MACHINES USE HARDWARE.

Can TLB and caching be overlapped?



This works, but …

Q. What is the downside?

A. Inflexibility. Size of cache limited by page size.

VA: 64bits

PA: 40bits

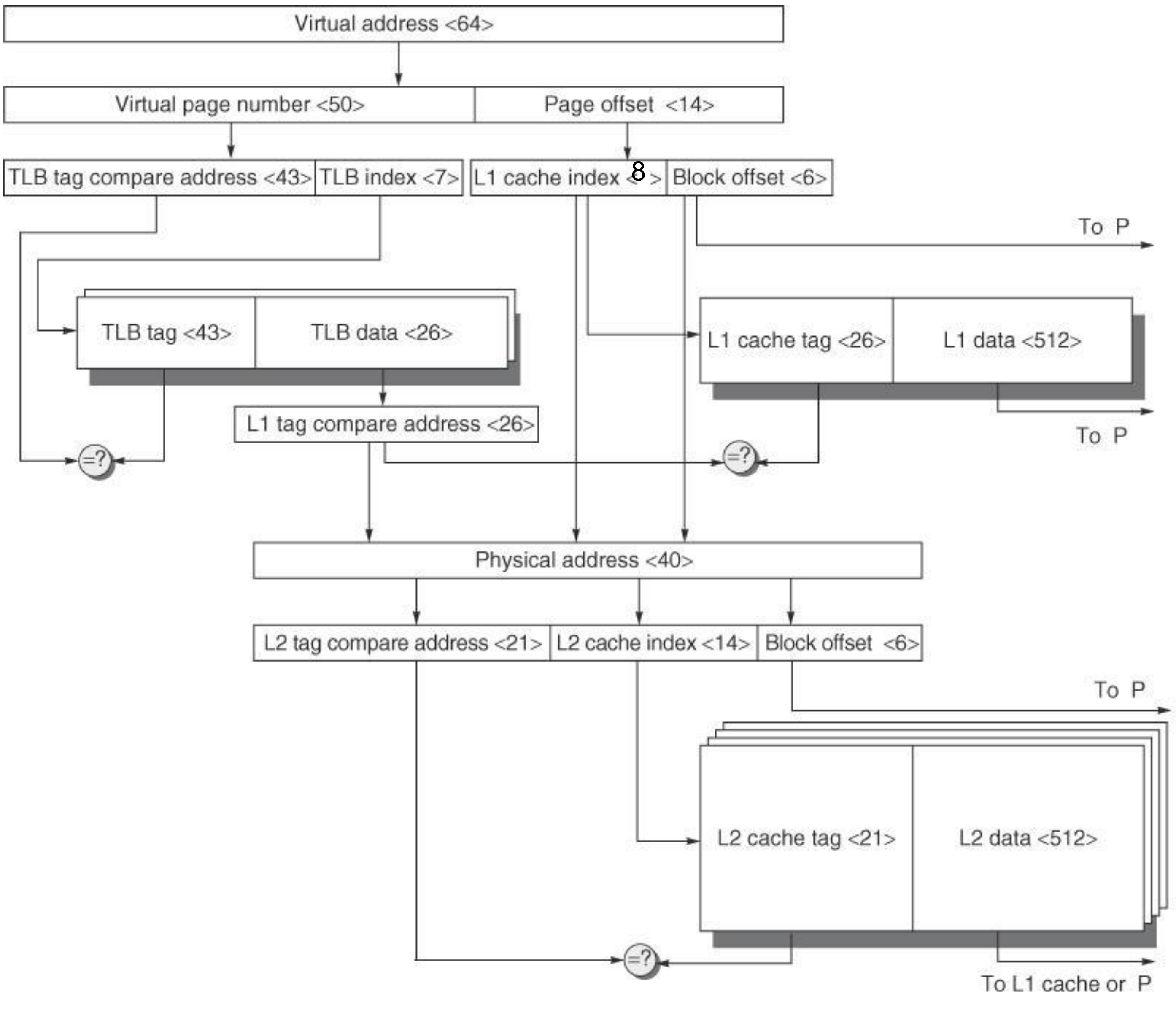
Page size: 16KB

TLB: 2-way set associative, 256 entries

Cache block: 64B

L1: direct-mapping, 16KB

L2: 4-way set associative, 4MB



Virtual Memory and Virtual Machines

- Protection via virtual memory
  - Keeps processes in their own memory space
- Role of architecture
  - Provide user mode and supervisor mode
  - Protect certain aspects of CPU state
  - Provide mechanisms for switching between user mode and supervisor mode
  - Provide mechanisms to limit memory accesses
  - Provide TLB to translate addresses

Virtual Machines

- Supports isolation and security
- Sharing a computer among many unrelated users
- Enabled by raw speed of processors, making the overhead more acceptable

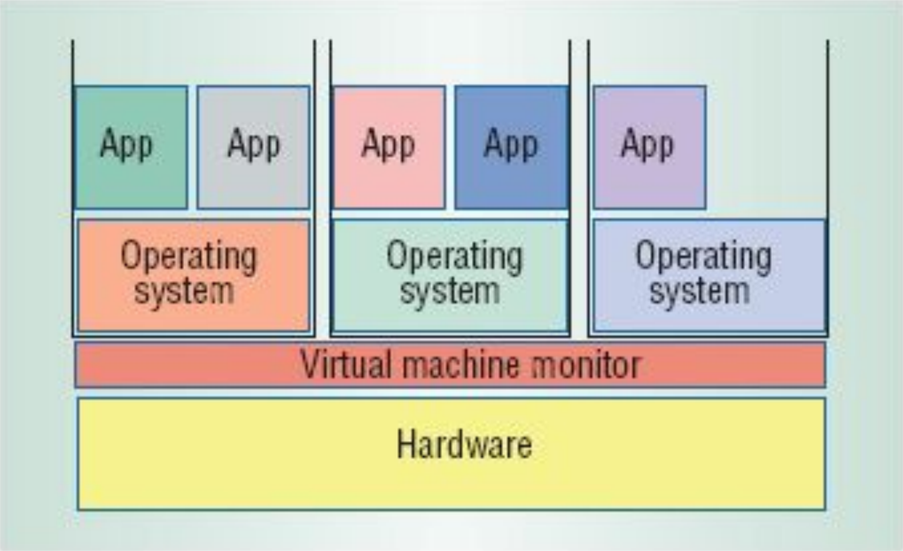
- Allows different ISAs and operating systems to be presented to user programs
  - "System Virtual Machines"
  - SVM software is called "virtual machine monitor" or "hypervisor"
  - Individual virtual machines run under the monitor are called "guest VMs"

Requirements of VMM

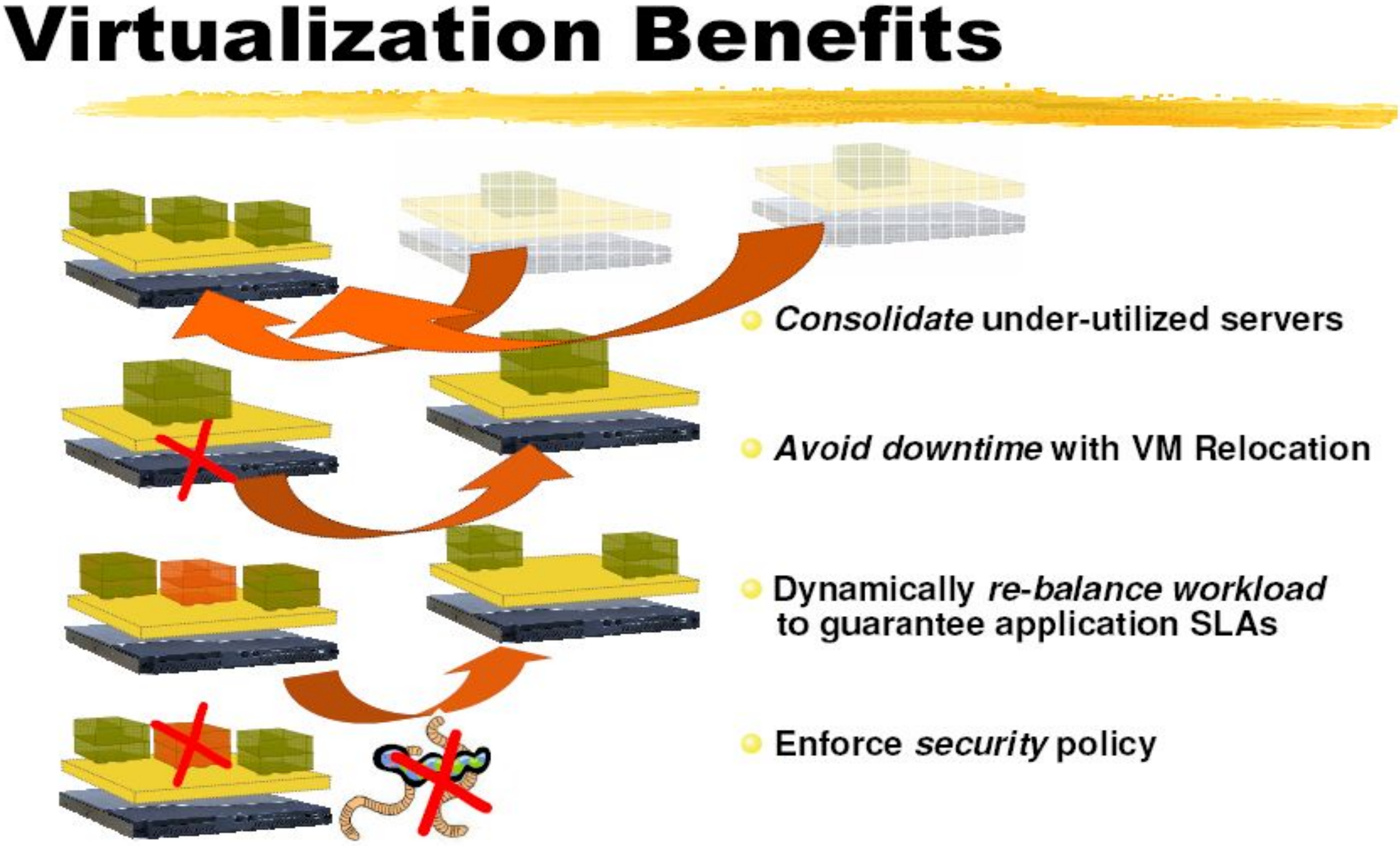
- Guest software should:
  - Behave on as if running on native hardware
  - Not be able to change allocation of real system resources
- VMM should be able to "context switch" guests
- Hardware must allow:
  - System and use processor modes
  - Privileged subset of instructions for allocating system resources

Virtual Machine Monitors (VMMs)

- Virtual machine monitor (VMM) or hypervisor is software that supports VMs
- VMM determines how to map virtual resources to physical resources
- Physical resource may be time-shared, partitioned, or emulated in software
- VMM is much smaller than a traditional OS;
  - isolation portion of a VMM is ≈ 10,000 lines of code



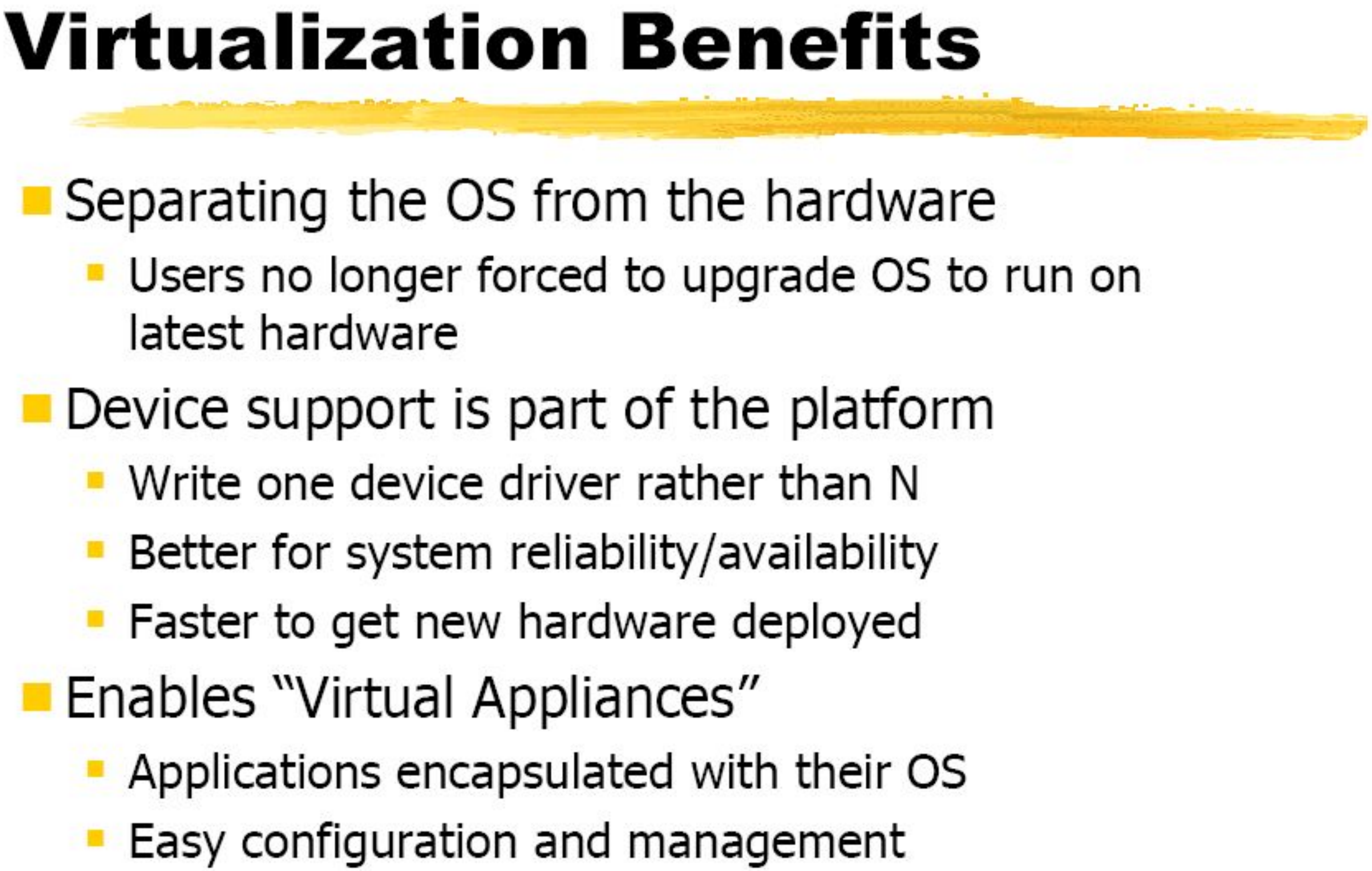
Virtual Machine Monitors (VMMs)



Virtualization Benefits

- *Consolidate* under-utilized servers
- *Avoid downtime* with VM Relocation
- Dynamically *re-balance workload* to guarantee application SLAs
- Enforce *security* policy

Virtual Machine Monitors (VMMs)



Virtualization Benefits

- Separating the OS from the hardware
  - Users no longer forced to upgrade OS to run on latest hardware
- Device support is part of the platform
  - Write one device driver rather than N
  - Better for system reliability/availability
  - Faster to get new hardware deployed
- Enables "Virtual Appliances"
  - Applications encapsulated with their OS
  - Easy configuration and management

Impact of VMs on Virtual Memory

- Each guest OS maintains its own set of page tables
  - VMM adds a level of memory between physical and virtual memory called “real memory”
  - VMM maintains shadow page table that maps guest virtual addresses to physical addresses
    - Requires VMM to detect guest’s changes to its own page table
    - Occurs naturally if accessing the page table pointer is a privileged operation

Extending the ISA for Virtualization

- Objectives:
  - Avoid flushing TLB
  - Use nested page tables instead of shadow page tables
  - Allow devices to use DMA to move data
  - Allow guest OS's to handle device interrupts
  - For security: allow programs to manage encrypted portions of code and data

Fallacies and Pitfalls

- Predicting cache performance of one program from another
- Simulating enough instructions to get accurate performance measures of the memory hierarchy
- Not deliverying high memory bandwidth in a cache-based system